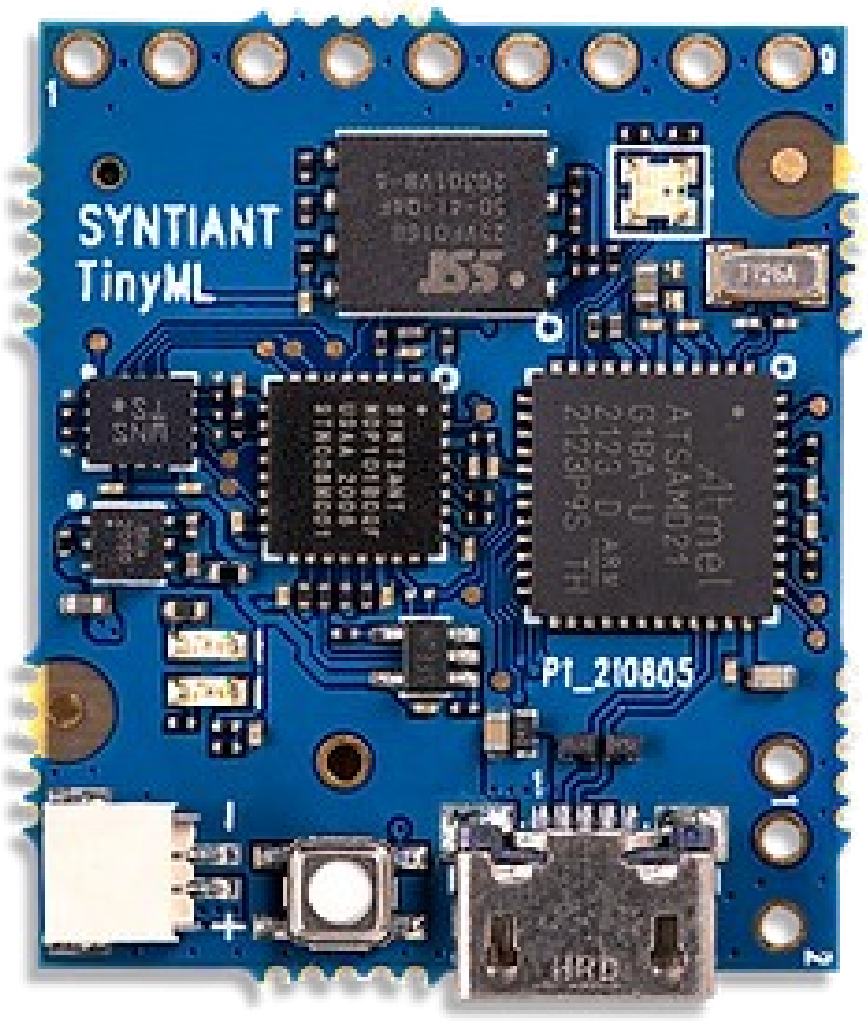
Taneli Heikkinen

Tekoälyn soveltaminen sulautetussa laiteympäristössä



Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2023

KAMK • University of Applied Sciences

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Taneli Heikkinen

Työn nimi: Tekoälyn soveltaminen sulautetussa laiteympäristössä

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: tekoäly, koneoppiminen, neuroverkot, sulautettu tekoäly, reunalaskenta, sulautetut järjestelmät, TinyML

Tämä opinnäytetyö tehtiin Kajaanin ammattikorkeakoulun tilauksesta. Sen keskeisiä aihepiirejä ovat tekoäly, koneoppiminen, neuroverkot, sulautettu tekoäly, reunalaskenta ja sulautetut järjestelmät. Opinnäytetyön tavoitteena oli tutkia sulautetun tekoälyn teoreettista pohjaa sekä alan tutkimuksen, menetelmien, ja markkinoilla olevien tuotteiden tilaa. Teoreettisen taustan tuli tarkastella ongelmia, jotka liittyvät tekoälyn toteutukseen resursseiltaan rajoitetuissa sulautetuissa laiteympäristössä, ja etsiä niihin ratkaisuja. Tavoitteena oli, että taustatyö soveltuu käytettäväksi opetusmateriaalina, ja auttaa sulautetun koneoppimisprojektin toteutuksessa. Työn käytännön osuudessa tuli toteuttaa tekoälyyn perustuva varashälytin, joka soveltaa käsiteltyjä tekniikoita.

Opinnäytetyöprosessissa tutkittiin tekoälyn ja koneoppimisen keskeisiä käsitteitä ja teoriaa. Työ tarkasteli aihetta sulautetun laiteympäristön kannalta ja keskittyi siinä toteutettavaksi soveltuviin teknologioihin. Siinä esiteltiin markkinoilla olevia kehitysalustoja ja ohjelmia, jotka tukevat sulautettua koneoppimista. Koneoppimisalgoritmien luokitteluperusteet määriteltiin, ja koneoppimisprojektin vaiheille esiteltiin kaksi mallia. Neuroverkoista käsiteltiin niiden rakenne ja sulautetun laiteympäristön kannalta oleellisimmat arkkitehtuurit.

Työn käytännön osuudessa toteutettiin koneoppimisprojekti soveltaen erityisiä menetelmiä ja laitteistoa, jotka on kehitetty optimoimaan koneoppimista sulautetuilla laitteilla. Projektissa käytettiin Syntiant TinyML -kehitysalustaa. Projektiosuutta varten suunniteltiin ja toteutettiin tarvittavan näyteaineiston keräys ja käsittely. Kerättyä näyteaineistoa käytettiin luokittelumallin harjoittamiseen Edge Studio -ohjelmassa käyttäen ohjattua opettamista. Prosessin tuloksena syntyi kehitysalustalla ajettava laiteohjelmisto, sekä tilastotietoa kehityn mallin toiminnasta. Mallin suorituskyvystä saatiin lukuja, jotka olivat kriteeri projektin onnistumista arvioitaessa.

Projektissa onnistuttiin osoittamaan käsiteltyjen tekniikoiden soveltuvuus koneoppimismallin toteutukseen käytännössä. Työn tavoitteena ollut varashälyttimen luokittelualgoritmi pystyttiin toteuttamaan ja ajamaan kohdealustalla. Sen suorituskyky arvioitiin luokittelun tilastotietojen perusteella täyttävän asetetut tavoitteet. Jatkokehitystä varten toteutettavaksi ominaisuudeksi työssä jäi signaaliketjun jatkaminen suunnitellulta laitteelta eteenpäin.

KAMK • University of Applied Sciences

Abstract

Author: Taneli Heikkinen

Title of the Publication: Implementing Artificial Intelligence in Embedded Systems

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: artificial intelligence, machine learning, neural networks, tiny machine learning, edge computing, embedded systems, TinyML

The thesis was commissioned by Kajaani University of Applied Sciences. Its main topics include artificial intelligence, machine learning, neural networks, embedded artificial intelligence, edge computing and embedded systems. The aim of the thesis was to examine the theoretical background of embedded artificial intelligence and the state of its research, methods, and products on the market. Problems related to the implementation of artificial intelligence in a resource-limited embedded device environment were to be studied, and possible solutions discovered. The goal was for the background work to be suitable for use as teaching material, and to help in the implementation of embedded machine learning projects. In the practical part of the thesis, a security alarm system based on artificial intelligence, and applying the discussed techniques, was to be implemented.

During the thesis process, the central concepts and theory of artificial intelligence and machine learning were studied. The topic was examined from the point of view of embedded device development and focused on technologies suitable for implementation this context. Development platforms and programs on the market that support embedded machine learning were examined. The classification criteria for machine learning algorithms were defined, and two models were presented for the stages of a machine learning project. The structure and the most essential architectures for neural networks in the embedded device environment were discussed.

In the practical part of the thesis a machine learning project was completed applying special methods and equipment developed for machine learning in embedded devices. The project used the Syntiant TinyML development platform. The collection and processing of the necessary dataset was planned and implemented. The collected sample data was used to train the classification model in the Edge Studio program using supervised learning. The product of the process was a firmware that can be run on the development platform, as well as statistical information about the operation of the developed model. Numerical data was obtained about the performance of the model, to be used as criteria for evaluating the success of the project.

The project succeeded in demonstrating the applicability of the discussed techniques for the practical implementation of a machine learning model. The security alarm classification model, which was the goal of the work, was successfully implemented and run on the target platform. Its performance was evaluated based on the statistical data of the classification model and was found to meet the target goals. For further development, the continuation of the signal chain from the device as a feature was discussed.

Sisällys

1 Johdanto

Tekoälyn edistys on ollut viime vuosina nopeaa, ja työkalut sen toteuttamiseksi ovat tulleet yhä laajemmin saataville. Vuonna 2011 Googlen Brain työryhmä aloitti työskentelyn syviin neuroverkkoihin perustuvan tekoälyn parissa. Yksi ensimmäisistä sovelluksista oli malli, joka kykeni tunnistamaan, onko kuvassa kissa vai ei. Tämän tehtävän opettamiseen tarvittiin kuusitoistatuhatta suoritinta luomaan yli miljardin kytköksen neuroverkko, joka työskenteli kolme päivää kymmenen miljoonan kuvan aineistolla. Tutkimuksen tuloksena syntyi myöhemmin suosittu avoimen lähdekoodin TensorFlow-ohjelmointiympäristö. [1.]

Tämän opinnäytetyön aiheena on sulautettu koneoppiminen (*engl. tiny machine learning*), kehittyvä reunalaskennan konsepti, joka yhdistää sulautetut järjestelmät ja koneoppimisen. Sen tavoitteena on toteuttaa koneoppimisen päättelykyky akulla toimivissa laitteissa erittäin matalalla virrankulutuksella, pienillä kustannuksilla, matalalla viiveellä ja paremmalla tietoturvalla. [3.] Pilvilaskennalla toteutettavien yleisten mallien sijaan sulautettu koneoppiminen keskittyy käyttäjälle yksilöityjen sovellusten toteuttamiseen [4 s. 1].

Tekoälymikropiirien markkinoiden arvioidaan kasvavan 42 % vuosien 2020 ja 2024 välillä. Joustavuutensa ansiosta sulautetulle tekoälylle löytyy sovelluskohteita laajasti aloilta, kuten kunnonvalvonta, poikkeamien havaitseminen, vikadiagnostiikka, älykäs kodin automaatio, valaistuksen ohjaus, energiansäästösovellukset, tuotantoketjun ennakointi, hahmontunnistus, konenäkö, neuroverkot robotiikan navigaatio ja puheentunnistus. [5.]

Tämän opinnäytetyön tavoite on kartoittaa sulautetun tekoälyn teoreettista pohjaa sekä alan tutkimuksen, menetelmien ja markkinoilla olevien tuotteiden tilaa. Työn sisältö suunniteltiin Kajaanin ammattikorkeakoulun tarpeita varten, ja siihen kuluu käytännön osuus, jossa toteutetaan sulautettu koneoppimisprojekti. Työn aihe on rajattu niihin tekoälyn osa-alueisiin, jotka on mahdollista toteuttaa sulautetussa ympäristössä. Projektin tarkoitus on osoittaa koneoppimismallin toteutuksen mahdollisuus resursseiltaan rajoitetulla kehitysalustalla hyödyntäen tutkittuja menetelmiä ja ohjelmistoja. Tavoitteena on, että työtä voidaan hyödyntää opetusmateriaalina, ja se auttaa vastaavien projektiopintojen toteuttamisessa.

2 Tekoäly ja koneoppiminen

Tekoäly on laaja tieteenala, joka John McCarthyn määritelmän mukaan tutkii älykkäiden koneiden, erityisesti tietokoneiden, käyttöä päätöksenteossa mielivaltaisesti määriteltyjen tavoitteiden saavuttamiseksi [6]. Tekoäly jaetaan usein vahvaan tekoälyyn, joka osoittaa yleistä ymmärrystä tai jopa tietoisuutta asioista, ja heikkoon tekoälyyn, joka toimii tietyssä rajoitetussa tehtävässä. Kaikki nykyiset tekoälyn sovellukset kuuluvat heikkoon tekoälyyn.

Koneoppiminen on tekoälyn osa-alue, jossa pyritään antamaan tietokoneelle kyky oppia tehtäviä ilman, että se nimenomaisesti ohjelmoidaan tekemään niitä. Siitä on tullut viime vuosina luultavasti tärkein tapa, jolla tekoälyä toteutetaan. Koneoppimisen sovelluskohteita ovat mm. hakukoneet ja suosittelualgoritmit, kuvien analysointi, lääketieteellinen diagnosointi ja itseajavat autot. [7.]

Syväoppiminen on koneoppimisen ala, joka käsittelee syvien neuroverkkomallien opetusalgoritmeja. Syväoppiminen on saavuttanut suurta menestystä sellaisilla alueilla kuin konenäkö, puheentunnistus ja luonnollisen kielen prosessointi. [4 s. 54.] Syväoppiminen on usein tekoälytaiteen ja keskustelubottien (esim. ChatGPT) taustalla [8].

2.1 Koneoppiminen

John D. Kelleherin mukaan koneoppimisessa tutkitaan algoritmeja, jotka löytävät funktioita tietoaineiston (engl. dataset) perusteella. Matematiikassa funktio on kuvaus syötearvojen joukosta ulostuloarvoon tai arvoihin. [9 s. 16.] Ohjelmaksi koodattuja funktioita kutsutaan koneoppimisessa malleiksi. Mallin toteutus voi olla tietokoneohjelma, laitteistotasolla toimiva piiri, keinotekoinen neuroverkko, tai fyysistä kytkentää simuloiva ohjelma. Koneoppiva algoritmi on prosessi, joka pyrkii löytämään mallin, joka parhaiten selittää tietoaineistossa esiintyvien piirteiden väliset suhteet. [9 s. 19.]

Prosessiin kuuluu yleensä kaksi vaihetta: opetus ja inferenssi eli päättely. Opetuksessa koneoppiva algoritmi käsittelee tietoaineistoa ja pyrkii löytämään sitä parhaiten vastaavan mallin. Päättelyssä löydettyä mallia sovelletaan uuteen, tuntemattomaan tietoaineistoon, josta tuotetaan arvioita ulostuloarvoille. [9 s. 21.] On olemassa oppimistyyppejä, kuten vahvistusoppiminen, jossa kahta vaihetta suoritetaan yhtä aikaa [10 s. 367].

Koneoppimisalgoritmit kykenevät oppimaan datasta [11 s. 97]. Tom Mitchellin määritelmä oppimiselle on, että ohjelman voidaan sanoa oppivan kokemuksesta E, kun sen suorituskyky tehtävässä T paranee kokemuksen E jälkeen arvioituna suorituskyvyn mitalla P. [10 s. 2.]

Määritelmän tarkoittamia tehtäviä voivat olla muun muassa luokittelu, regressio, transkriptio, ryhmittely, kääntäminen, poikkeaman havaitseminen, kohinan poisto, tai synteesi ja näytteistys. Tehtäviä voidaan kutsua myös koneoppimisongelmiksi. [11 s. 97.]

Koneoppimisalgoritmin kokemus voidaan määritellä sen perusteella mitä oppimistyyppiä käytetään, ja mitä tietoaineistoa algoritmille annetaan [11 s. 102]. Suorituskyvyn mitta on tehtäväkohtainen, mutta se on jokin numeerisesti ilmaistava arvo. Arvon antaa neuroverkolle määriteltävä hukkafunktio. Luokittelualgoritmille se voi olla esimerkiksi osuus syötteistä, joille malli tuottaa oikean ulostulon. Suorituskyvylle voi olla vaikea löytää arviointikriteeriä, joka vastaa systeemiltä haluttua käyttäytymistä. [11 s. 101.]

2.1.1 Oppimistyypit

Koneoppimisalgoritmit voidaan luokitella niille annettavan opetusdatan luonteen perusteella. Ohjattu oppiminen on yleisin oppimistyyppi. Siinä opetusdata on nimettyä, eli jokaiselle syötteelle annetaan ennalta tunniste, johon opetettavan mallin halutaan päätyvän. Mallin annetaan tehdä ennustuksia harjoitusdatalla, ja sen toimintaa korjataan, kun ennusteet ovat vääriä. Opetus jatkuu, kunnes algoritmi saavuttaa opetusdatalla halutun tarkkuuden. [12.]

Ohjaamattomassa oppimisessa opetusdataa ei ole nimetty valmiiksi, eikä ulostuloa tiedetä ennalta. Algoritmin tehtävä on löytää opetusdatassa rakenteita, tai päätellä säännönmukaisuuksia. [12.] Se yrittää tunnistaa mallin, joka parhaiten jakaa syötteet pääteltyihin rypäisiin eli klustereihin syötteiden samankaltaisuuksien perusteella [9 s. 33]. Puoliohjatussa oppimisessa tietoaineisto on sekoitus luokiteltua ja luokittelematonta dataa. On olemassa haluttu lopputulos, mutta mallin täytyy oppia organisoimaan dataa luokittelun lisäksi. [12.]

Vahvistusoppimiselle on tyypillistä, että ei ole olemassa rajattua tietoaineistoa, vaan malli on vuorovaikutuksessa ympäristönsä kanssa. Jokaista herätettä kohti ei ole olemassa tiedettyä oikeaa ulostuloa. Sen sijaan harjoitettavan mallin tehtävä on oppia epäsuorasta, viiveellä tulevasta pa-

lautteesta valitsemaan sarja toimintoja, joka tuottaa suurimman kumulatiivisen palkinnon. Mallilla on funktio, joka arvioi odotettua palkintoa nykyisen tilanteen perusteella. Tätä oppimismallia käytetään paljon robotiikassa ja peleissä. [10 s. 367.]

2.1.2 Koneoppimisongelmat

Koneoppimista voidaan soveltaa luokitteluun, jossa algoritmin tehtävä on määritellä mihin kategoriaan jokin syöte kuuluu. Luokkia on tunnettu määrä. Ongelman ratkaisemiseksi algoritmin tulee tuottaa funktio, joka antaa jokaiselle vektorina esitetylle syötteelle luokan ilmoittavan numeerisen ulostuloarvon. Esimerkki luokittelutehtävästä on hahmontunnistus, missä syöte on kuva numeerisesti kuvattuina pikseliarvoina. Ulostulo on koodi, joka osoittaa kuvassa esiintyvän hahmon. Hahmontunnistus voidaan toteuttaa tehokkaasti syväoppimista hyödyntäen. [11 s. 98.]

Luokittelutehtävä on haastavampi, jos syötevektorissa ei aina ole kaikkia arvoja. Tällöin luokittelualgoritmin tulee oppia useita funktioita. Jokainen funktio vastaa luokittelua erilaisella syötevektorin muunnoksella. Suuren funktiomäärän määrittelemiseksi on mahdollista käyttää tilastollisia menetelmiä. [11 s. 98.]

Regressiotehtävässä algoritmin täytyy ennustaa numeerinen arvo syötteen perusteella. Ratkaisu on funktio, mutta luokittelusta poiketen ulostuloarvo ei kuulu rajattuun määrään luokkia. Esimerkki regressiosta on tulevien hintojen ennustaminen arvopapereille. [11 s. 99.]

Regressioalgoritmeja ovat esimerkiksi lineaarinen regressio ja logistinen regressio.

Transkriptiossa tehtävä on tarkastella syötedataa, ja antaa sen sisällöstä kuvaus diskreetissä tekstimuodossa. Esimerkiksi Google Street View -ohjelma käyttää syväoppimista osoitenumerojen poimimiseen katukuvista tekstimuotoon. [11 s. 98.]

Poikkeaman tunnistaminen tarkoittaa tapahtumia tai objekteja kuvaavan syötedatan läpikäymistä, josta liputetaan epätyypilliset näytteet. Tehtävä on tavallinen esimerkiksi kunnonvalvonnassa vikojen havaitsemiseksi laitteissa. [11 s. 100.]

Ryhmittely (*engl. clustering*) on luokittelua muistuttava tehtävä, mutta luokkien määrä on ennalta tuntematon. Menetelmässä syötteet ryhmitellään rypäisiin niin, että eniten samankaltaisuuksia

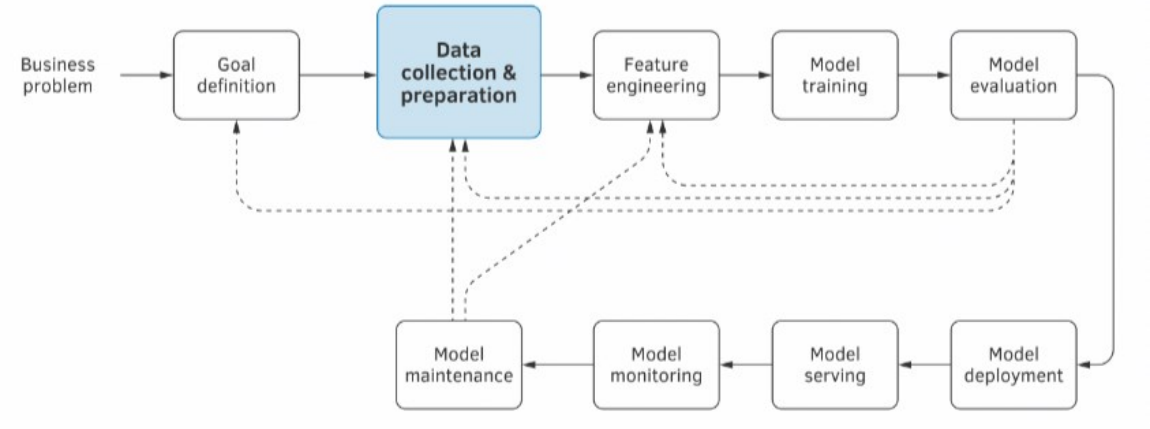
keskenään jakavat näytteet päätyvät samaan rypääseen. Samankaltaisuuden kriteeri riippuu ryhmittelyalgoritmista. Niitä ovat mm. datan jakaumaan perustuvat algoritmit (*engl. distribution-based methods*) ja datan painopisteeseen perustuvat algoritmit (*engl. centroid-based methods*). [13.]

2.1.3 Koneoppimismallin suunnitteluprosessi

Andriy Burkovin esittämät vaiheet koneoppimisprojektille ovat:

1) Tavoitteiden määrittely
2) Datan keräys ja valmistelu
3) Piirteiden suunnittelu
4) Mallin harjoittaminen
5) Mallin arviointi
6) Mallin käyttöönotto
7) Mallin tarjoilu
8) Mallin valvonta
9) Mallin huolto [14.]

Projektin kaavio on esitetty kuvassa 1. [15 s. 19.]



Kuva 1. Koneoppimisprojektin vaiheet [14]

Malli on esitetty jatkuvana prosessina, jossa voidaan palata aiempiin vaiheisiin ominaisuuksien päivittämiseksi [15 s. 19]. Jotkin vaiheet voivat olla automatisoituja, esimerkiksi vahvistusoppimisessa ei erikseen kerätä tietoaineistoa, vaan opetus tapahtuu toimintaympäristössä. Syväoppimisessa piirteiden erottamista ei suorita suunnittelija, vaan neuroverkko kykenee tunnistamaan piirteet automaattisesti. [10 s. 369.]

Mallin tarjoilulla (*engl. model serving*) tarkoitetaan käyttötapausta, jossa käyttöönotetulle mallille annetaan syötteitä ihmisen tai koneen antamien pyyntöjen perusteella ja palautetaan vasteet. [16 s. 3.]

Raul V. Rodriguez esittää vastaavan prosessin hieman eri tavalla jaettuna:

1) Tavoitteiden määrittely
2) Tiedonkeruu
3) Datan valmistelu
4) Koneoppimismallin valinta
5) Mallin harjoittaminen
6) Arviointi
7) Parametrien hienosäätö
8) Mallin käyttöönotto [17.]

Ensimmäinen askel projektissa on ratkaistavan ongelman ja tavoitteiden määrittely. Arvioidaan projektin monimutkaisuus, tarvittavan datan määrä, mallilta vaadittavat ominaisuudet, ja onko niiden saavuttaminen mahdollista käytännössä. [17.]

Tiedonkeruuvaiheessa tutkitaan ja hankitaan tietoaineistoa, jota käytetään koneoppimismallin harjoittamiseen. Kerätyn datan laatu on tärkeää, koska se vaikuttaa suoraan, kuinka hyvin malli tulee toimimaan. Data voi tulla valmiista tietokannasta, tai se voidaan hankkia itse. [17.] Data voidaan hankkia näytteenotolla, jolloin otetaan näytteitä signaaleista, jotka mittaavat todellisia fysikaalisia ilmiöitä, ja muutetaan ne digitaaliseen numeeriseen muotoon arvoiksi, joita tietokone voi käsitellä. Data tallennetaan, ja kerätään käyttöpaikkaan mahdollisesti useista lähteistä. Data on yleensä raakadataa, joka ei sovellu sellaisenaan välittömään käyttöön. [18.]

Datan valmisteluvaiheeseen kuuluu sen siistiminen, yhtenäistäminen ja nimeäminen. Nimeäminen tehdään ohjatun oppimisen tapauksessa. Jos mallin tehtävä on luokittelu, nimetään harjoitusdatan luokat. Piirteidensuunnitteluvaiheessa (*engl. feature engineering*) luodaan piirteet raakadatasta, jos kyseessä ei ole piirteet itsenäisesti poimiva syväoppimismalli. [18.] Datan määrä tasapainotetaan eri tunnisteiden kesken, jotta tulokset eivät keskity tietylle tunnisteelle. Lisäksi

data jaetaan harjoitusdataan ja arviointidataan, jota käytetään myöhemmin mallin suorituskyvyn arviointiin, ja ylisovittamisen välttämiseen. Tyypillinen jako on 80 % harjoitusdatan hyväksi. Tässä vaiheessa dataa voidaan esikäsitellä normalisoimalla, kopioiden poistolla tai virheiden korjauksella. [17.]

Käytettävä koneoppimismalli valitaan sen perusteella mikä ratkaistava ongelma on (esim. luokittelu, regressio tai syväoppiminen), ja minkä tyyppistä dataa käytetään (esim. kuva, ääni tai teksti). Koneoppimismallin opetusvaihetta varten asetetaan hyperparametrit, jotka vaikuttavat mm. oppimisnopeuteen, ja opetusprosessi käynnistetään. Valittu algoritmi säätää mallin painoarvoja, ja sen suorituskyky paranee asteittain. Prosessin tuloksena saadaan koneoppimismalli. [17.]

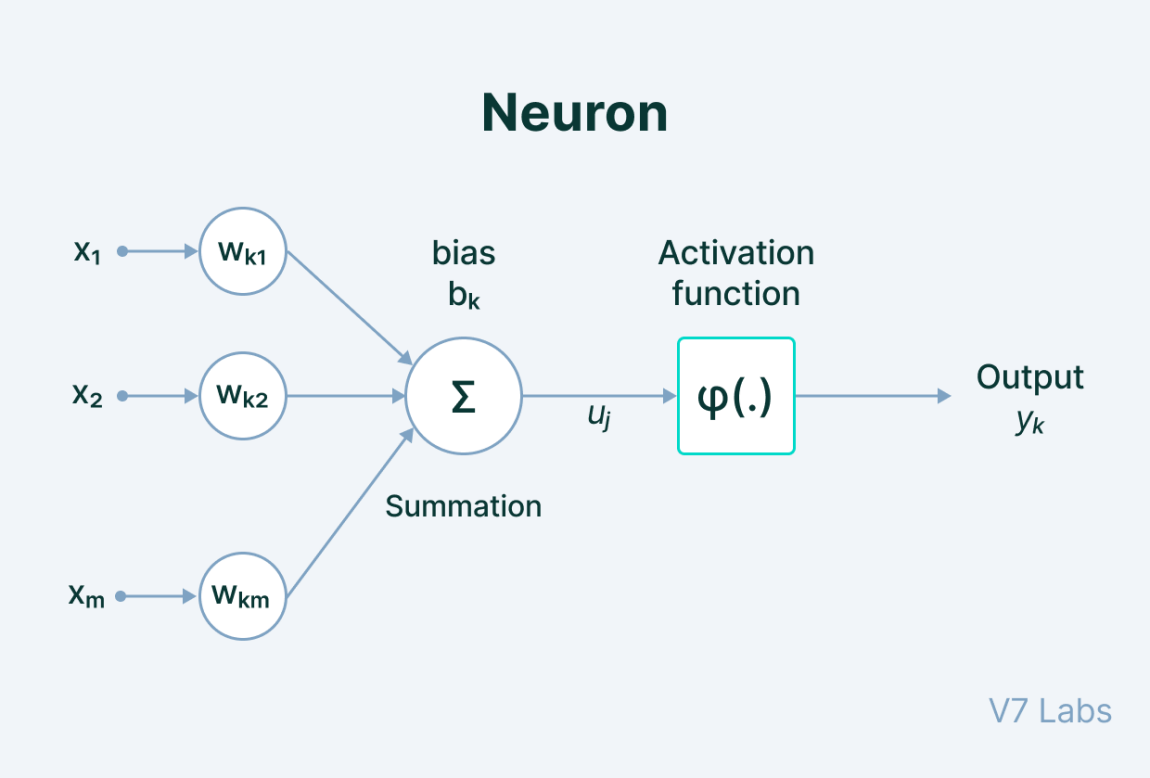
Arviointivaiheessa käytetään aiemmin erilleen asetettua testausdataa mallin suorituskyvyn arviointiin. Käyttämällä mallille ennen näkemätöntä dataa pyritään varmistamaan, ettei sitä ole ylisovitettu harjoitusdataan. Jos arviointi ei tuota tyydyttäviä tuloksia, on mahdollista palata takaisin oppimisvaiheeseen ja säätää sen parametreja. Parametrien hienosäätövaiheessa voidaan muuttaa esim. iteraatioiden määrää tai oppimistahtia (*engl. learning rate*), joka määrää muutosten määrän painoarvoissa iteraatioiden välillä, ja käytännössä vaikuttaa oppimisaikaan. Hyperparametrit vaihtelevat käytettävän algoritmin mukaan. Tarvittaessa voidaan palata vielä aikaisempiin vaiheisiin prosessissa. [17.]

Kun mallin suorituskyky on riittävä, opetettu koneoppimismalli ladataan käyttöönottovaiheessa kohdealustalle (jos se opetettiin eri laitteessa), ja sitä sovelletaan päätöksentekoon käytännössä [17].

2.2 Neuroverkot

Keinotekoinen neuroverkko koostuu yksinkertaisista tietoa käsittelevistä yksiköistä, neuroneista. Neuroni on esitetty kuvassa 2, jossa informaatio virtaa vasemmalta oikealle. Neuronilla on useita sisääntuloja, ja yksi lähtö. Syötteet x_m ovat numeerisia arvoja, ja niihin vaikuttavat kertoimet,

joita kutsutaan neuronin painoiksi w_{km}. Kytkennän painot vaikuttavat siihen, miten neuroni käsittelee saamaansa tietoa, ja neuroverkon harjoittaminen perustuu painojen säätämiseen. Jokainen tulo ja lähtö on mahdollista yhdistää toiseen neuroniin. Syötteille lasketaan painotettu summa u_j, joka viedään aktivaatiofunktiolle. Se kuvaa neuronin lopullisen ulostulon y_k numeerisena arvona. [9 s. 66.] Aktivaatiofunktio mahdollistaa epälineaarisen suhteen neuroverkon tulon ja lähdön välillä [9 s. 74].

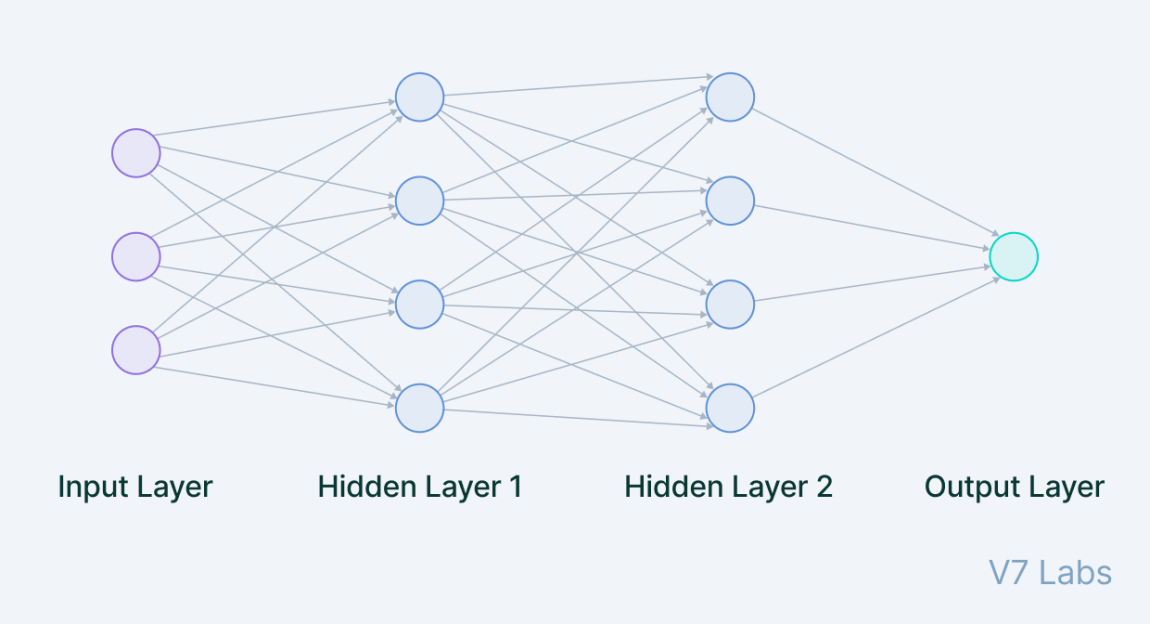


Kuva 2. Neuroni [19]

Kun useita neuroneita kootaan kerroksiksi, ne muodostavat neuroverkon. Neuroverkkojen tyyppejä ovat mm. eteenpäin kytketty (monikerroksinen) neuroverkko (*engl. multilayer perceptron*, *MLP*), konvoluutioneuroverkko (*engl. convolutional neural network, CNN*) ja takaisinkytketty neuroverkko (*engl. recurrent neural network*, *RNN*). Monet neuroverkkorakenteet, kuten RNN ja LSTM (*long short-term memory*) ovat saavuttaneet merkittävää edistystä esimerkiksi luonnollisen kielen käsittelyssä, mutta ovat laskennallisesti raskaita. [4 s. 25.]

2.2.1 Eteenpäin kytketyt neuroverkot

Yleisin neuroverkkojen arkkitehtuuri on eteenpäin kytketty verkko. Siinä yhden kerroksen jokainen neuroni on yhdistetty seuraavan kerroksen jokaiseen neuroniin, mikä on esitetty kuvassa 3. Sitä voidaan kutsua myös täysin yhdistetyksi verkoksi. Tulokerroksen ja ulostulokerroksen välissä ovat piilokerrokset. Täysin yhdistettyjä kerroksia voi olla myös osina muiden arkkitehtuurien neuroverkoissa. Neuroverkkoa kutsutaan syväksi, jos se sisältää tietyn määrän monimutkaisuutta, yleensä enemmän kuin kaksi piilokerrosta. [11 s. 165.]



Kuva 3. Eteenpäin kytketty neuroverkko [19]

Neuroverkon syötteet ja painot voidaan kuvata matriiseina. Matriisin yksi alkio voi sisältää esimerkiksi kuvan yhden pikselin informaation numeerisena arvona. Jokaisella verkon kerroksella on sisääntulomatriisi ja ulostulomatriisi. Suhde verkon yhden kerroksen tulojen ja lähtöjen välillä voidaan ilmaista matriisin ja tulovektorin tulona, joka on esitetty kaavassa (1), missä u on ulostulojen vektori, W on kerroksen painojen matriisi, x on sisääntulojen vektori ja b ovat mahdollisten biasarvojen vektori. Yhtälö viittaa myös MAC (*multiply–accumulate*) operaatioon[4 s. 25].

$$u = Wx + b \tag{1}$$

Neuroverkkojen rakennetta kuvaavista tietorakenteista käytetään usein termiä tensori. Se on yleistys, johon kuuluu muitakin kuin kaksiulotteisia matriiseja. Tensori on moniulotteinen taulukko ja matriisi on toisen asteen tensorin erityistapaus. [15 s. 10.]

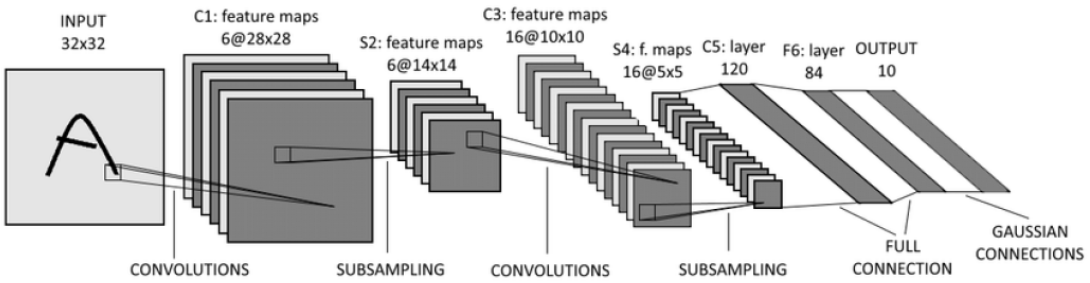
Mitä enemmän dataa neuroverkolle halutaan syöttää, sitä suuremmaksi kasvaa vaadittavien neuronien määrä. Perinteiset neuroverkkoarkkitehtuurit voivat tuottaa halutun tuloksen, mutta ne ovat laskennallisesti ja muistin kannalta raskaita. Monimutkaisten neuroverkkojen ajaminen resursseiltaan rajoitetuilla alustoilla vaatii käytetyn muistin vähentämistä mallien tiivistämisellä, tai soveltamalla laitteistokiihdytystä sulautetuilla järjestelmillä. [4.]

2.2.2 Konvoluutioneuroverkot

Konvoluutioneuroverkot suunniteltiin alun perin kuvantunnistustehtäviin, vaikka ne eivät rajoitu pelkästään siihen. Ne pyrkivät ratkaisemaan suuren syötedatamäärän ongelman tiivistämällä dataa usealla menetelmällä. Kuvantunnistuksessa ne ottavat vain pienen alueen kuvasta kerrallaan käsittelyyn. Sen sijaan että kuvan jokainen pikseli olisi syötteenä neuroneille, etsitään syötteiksi kuvan paikallisten alueiden piirteet. Piirteet ovat syötedatan osa-alueen kuvaus, joka ei sisällä kaikkea sen informaatiota. [20 s. 41.]

Tyypillinen konvoluutioneuroverkko koostuu kolmesta kerrostyypistä: konvoluutiokerroksista, koontikerroksista (*engl. pooling layer*) ja eteenpäin kytketyistä kerroksista.[4 s. 34.] Sulautetun tekoälyn konvoluutioneuroverkot keskittyvät kevytrakenteisiin konvoluutiosuodattimiin [4 s. 26].

Kuvassa 4 on esitetty esimerkki konvoluutioneuroverkon (LeNet-5) rakenteesta. Herätteenä on kuva näytteistettynä matriisimuotoon. Tulossa ensimmäisenä on konvoluutiokerros, joka tuottaa syötematriisista pienemmän piirrekartan. Aktivaatiokerrokset on jätetty pois kuvan yksinkertaistamiseksi. Seuraavana on koontikerros, joka pienentää matriisia edelleen jakamalla sen osiin, ja korvaamalla kunkin osan sen alkioiden maksimiarvolla (tai keskiarvolla riippuen algoritmista). Koontikerroksen jälkeen seuraa uusi konvoluutiokerros, ja uusi koontikerros, jonka jälkeen tulevat eteenpäin kytketyt kerrokset. Ne suorittavat luokittelun supistetun piirrekartan perusteella. [4 s. 34.]



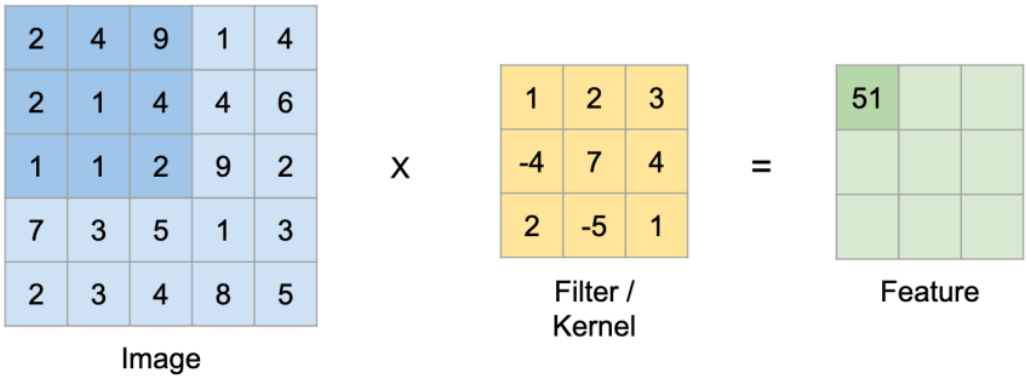
Kuva 4. Konvoluutioneuroverkon rakenne [4 s. 34]

Konvoluutiokerrokset koostuvat suodattimista, jotka ottavat kerrallaan käsittelyyn pienen alueen syötedatan matriisista. Suodatin, eli kernel-matriisi, on joukko painoarvoja, jotka kerrotaan sisääntulomatriisin osan kanssa alkioittain. Tulot summataan, tai niistä voidaan ottaa keskiarvo. Tämä operaatio on matriisien konvoluutio, ja sen tuloksena saadaan syötedatasta piirteet, joita käytetään sisääntulona neuroverkon seuraaville kerroksille. Neuroverkko suorittaa matemaattisen operaation käytännössä neuronien painoarvojen perusteella, ja miten sisääntulot on kytketty neuroneille edeltävältä kerrokselta. [4 s. 34.]

Kaavassa (2) on esitetty matriisien konvoluutio matemaattisesti. X on sisääntulomatriisi, X’ on ulostulomatriisi, K on kernel-matriisi ja b on biasarvo. Tässä tapauksessa operaattori ei ole matriisien kertominen, vaan konvoluutio-operaatio, eli matriisien vastaavat elementit kerrotaan keskenään ja summataan. [4 s. 34.]

$$X' = K * X + b \tag{2}$$

Kuvassa Kuva 5 on havainnollistettu konvoluution laskemista. Sisääntulomatriisin ja kernel-matriisin konvoluutio tuottaa yhden alkion piirrematriisiin. Loput alkiot saadaan laskemalla konvoluutiot muilta alueilta siirtämällä reseptiivistä kenttää, joka on merkitty kuvaan tummansinisellä.



Kuva 5. Konvoluutio [4 s. 35]

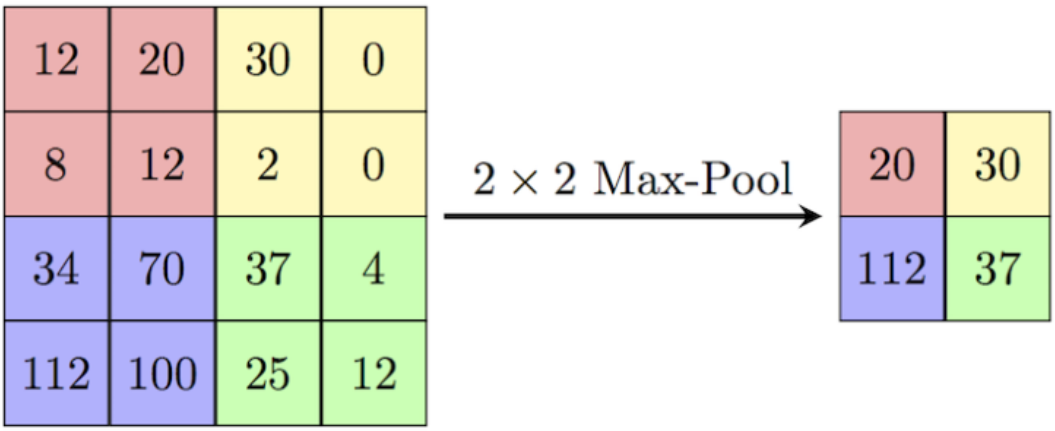
Neuronin reseptiivinen kenttä on alue, jolta sen syötteet tulevat sisääntulomatriisista. Alue, joilla kahden naapurineuronin reseptiiviset kentät ovat päällekkäin, on niiden peittymä. Peittymän suuruutta säädetään hyperparametrilla, jonka nimi on askelpituus (*engl. stride*). Kahden konvoluutio-operaation välillä siis siirretään reseptiivistä kenttää eteenpäin askelpituuden osoittama määrä alkioita. Konvoluutioprosessin tulosta sanotaan piirrekartaksi. [11 s. 330.]

Konvoluutio-operaatio ei sisällä aktivaatiofunktiota, vaan se suoritetaan piirrekartalle. Konvoluutiokerroksia seuraa yleensä aina epälineaaristen funktioiden kerros, eli aktivaatiokerros. Aktivaatiofunktiot eivät muuta matriisin muotoa, vaan kartoittavat sen alkioille uudet arvot.[4 s. 35.]

Yleisin aktivaatiofunktio konvoluutioneuroverkoissa on askelfunktio (*engl. rectified linear unit*). Se suoritetaan matriisille alkioittain. Funktio on esitetty kaavassa (3). [4 s. 36.].

$$ReLU(x) = max(0, x) \tag{3}$$

Koontikerrokset (*engl. pooling layers*) ovat toinen menetelmä datan supistamiseen. Niitä käytetään yleensä konvoluutiokerrosten jälkeen, jotta neuroverkolla olisi vähemmän sisääntuloparametreja. Sisääntulomatriisin osasta otetaan maksimiarvo tai keskiarvo, ja siitä muodostetaan alkio uuteen, pienempään matriisiin. Se syötetään eteenpäin seuraavalle kerrokselle. Operaatio on esitetty kuvassa 6. Koontikerroksen parametreja ovat ikkunan koko ja askel. Nämä määräävät miten suuresta matriisin alueesta koonti suoritetaan kerrallaan, ja miten paljon alueet menevät limittäin. Parametrit toimivat samalla periaatteella, kuin konvoluutiokerroksissa. Parametrit vaikuttavat ulostulomatriisin kokoon.[4 s. 35.]



Kuva 6. Koontioperaatio [4 s. 36]

Padding-operaatiossa lisätään sisääntulomatriisin reunoille alkioita, jotta sitä saadaan kasvatettua ennen konvoluutiota. Voidaan käyttää zero padding -menetelmää, jolloin alkioihin tulee nollia, tai near padding -menetelmää, jolloin niihin sijoitetaan lähin arvo matriisista. Näin voidaan säätää, saadaanko ulostulomatriisin koko parillisena. Konvoluutio saattaa alkaa niin, että kernelmatriisin keskusta asetetaan sisääntulomatriisin ensimmäisen alkion päälle. Tällöin osa kernelistä on matriisin ulkopuolella. Voidaan myös haluta, että matriisin koko ei pienene liian nopeasti neuroverkon seuraaville kerroksille mentäessä. Padding- ja stride-parametrit vaikuttavat ulostulomatriisin kokoon. [21.]

3 Sulautettu tekoäly

Tekoälysovellukset on usein sijoitettu pilveen, jossa ne voivat hyödyntää tehokasta rinnakkaista laskentaa. Pilvilaskennan haittapuolia ovat riippuvuus kalliista palvelimista, viive päätöksenteon tulosten noutamisessa, henkilökohtaisen tiedon paljastuminen kolmannelle osapuolelle ja rajattu mahdollisuus toteuttaa yksilöityjä sovelluksia. [4 s. 11.]

Sulautetun koneoppimisen etuja ovat matalampi viive, tehokas kaistan käyttö datalle, parempi tietoturva ja matalammat kustannukset. Sen avulla IoT-järjestelmät eivät ole jatkuvasti riippuvaisia pilvipalvelusta. Mahdollisia sovelluskohteita ovat mm. kulutuselektroniikka, autonomiset järjestelmät, terveydenhuolto, hajautettu tietojenkäsittely ja älykäs maatalous. [22.] Sulautettu koneoppiminen mahdollistaa datan keräyksen, käsittelyn ja tulkinnan lähellä sen lähdettä [3.].

Käytetyt laitteet ovat yleensä pieniä edullisia ja energiaa säästäviä (n. 1 mW). Esimerkiksi mikrokontrollerit 32-bittisestä STM32-tuoteperheestä, jotka perustuvat ARM Cortex-M-prosessoreihin, tukevat pieniä projekteja. Mikrokontrollereissa on prosessori, sulautettua flash-muistia ohjelmaa varten, SRAM-muistia datalle sekä sisään- ja ulostulorajapinnat. Cortex-M-prosessorit suorittavat yhden operaation kerrallaan, mikä on optimaalista matalan viiveen, kustannusten ja tehon kannalta. [3.]

3.1 Sulautettu koneoppiminen

Kehitys prosessorien arkkitehtuurissa on mahdollistanut kehittyneiden koneoppimisalgoritmien suorittamisen hyvin pienillä mikrokontrollereilla. Sulautetulla koneoppimisella tarkoitetaan alaa, jossa koneoppimisen laitteistoa, algoritmeja ja ohjelmistoja käytetään suorittamaan paikan päällä tapahtuvaa sensoridatan analyysiä hyvin matalalla virrankulutuksella, tyypillisesti alle milliwatin luokassa. Tämä mahdollistaa jatkuvasti päällä olevat paristokäyttöiset sovellukset. [5]

Gartner-yhtiön määritelmän mukaan sulautettu tekoäly tarkoittaa tekoälyn tai koneoppimisen tekniikoiden käyttöä sulautetuissa järjestelmissä paikallisesti tallennetun datan analysointiin. Kun teknologia yhdistetään reunalaskentaan, pystyään vähentämään viivettä, mahdollistamaan syvempi ennustava analyysi ja vähentämään vasteaikaa vakavissa vikatilanteissa. Aiheeseen liittyvä

termi on *Fog computing*, jossa laskenta sijaitsee reunalaskennan ja pilvilaskennan välissä lähiverkossa. [5]

Kun datan tuottamisen ja tallentamisen tahti on kasvanut nopeasti, on tullut yhä tärkeämmäksi, että on olemassa älykkäitä reunalaskennan laitteita, jotka kykenevät suorittamaan mukautuvaa ja inkrementaalista laskentaa määrättyihin tehtäviin. Datan määrän kasvaminen tarkoittaa laskennallisesti intensiivisempien AI-mallien kehitystä. [5]

Tyypillisesti sulautetuissa tekoälysovelluksissa laskennan ja muistin suhteen raskaat osuudet, kuten tekoälyn harjoittaminen, siirretään tehokkaille prosessoreille pilvessä, ja vain tietyt kevyet mallit ajetaan reunalaskennassa mobiili- ja IoT-laitteilla. Sulautettu tekoäly voi vähentää prosessien suoritusaikaa tarjoamalla tarkkoja päätelmiä tallennetusta datasta säilyttäen silti tietoturvan. [5]

3.1.1 Laitteistovaatimukset

Sulautetun tekoälyn laitteisto voidaan karkeasti jakaa yleistä laskentaa suorittavaan ja tiettyä tehtävää varten suunniteltuun elektroniikkaan. Edellinen voidaan edelleen jakaa latenssiin pohjautuviin prosessoreihin ja suorituskykyyn perustuviin prosessoreihin. Tehtäväpohjaisiin laitteisiin kuuluvat FPGA-, ja ASIC-piirit. [5]

Teollisia sovelluksia, kuten robotteja ja autonomisia ajoneuvoja, varten oleellisia vaatimuksia ovat matala viive ja virrankulutus. Näissä laitteissa sensorien täytyy reagoida välittömästi turvallisuuden ja autonomian takia. Perinteisiä tietokonearkkitehtuureja ei ole suunniteltu tätä silmällä pitäen. Vaatimukset edellyttävät datan siirtämistä muistin ja prosessorin välillä, mikä aiheuttaa korkean latenssin ja virrankulutuksen. Se tekee nämä arkkitehtuurit huonosti sulautetun tekoälyn sovelluksiin sopiviksi. [5]

Sulautetun tekoälyn laitteet koostuvat yleiskäyttöön tarkoitetuista prosessoreista ja graafisista laskentayksiköistä, sekä FPGA- ja ASIC-mikropiireistä. Jos käytetään tietyn toiminnon suorittamiseen tehtyjä erikoiskomponentteja, on kyse laitteistokiihdytyksestä. Se voi vähentää latenssia ja lisätä suorituskykyä, mikä tekee AI-algoritmeista, kuten konvoluutioneuroverkoista, sopivia kohteita FPGA-pohjaiseen laskentaan. Nämä AI-mallien mutkikkuudet, joihin liittyyvät ohjelmiston ja laitteiston yhteissuunnittelu, ajavat tarvetta laitteistokiihdytykselle, joka tukee valmiiksi AI mallien päättelykykyä ja koulutusta. [5]

3.1.2 Haasteet koneoppimismallien suunnittelussa

Harjoitusvaiheessa neuroverkot oppivat tuottamaan halutut tulokset muuttamalla neuronien paino- ja bias-arvoja. Harjoitettua mallia käytetään sitten ratkomaan ongelmia aidossa käyttöympäristössä. Prosessin nimi on *inference* eli päätösten tekeminen. [5]

Monimutkaisten algoritmien suoritus laitteissa, joissa on rajattu kapasiteetti, vaikuttaa niiden suunniteluun, latenssiin ja virrankulutukseen. Neuroverkkojen ajaminen rajatuilla resursseilla vaatii yhteistyötä algoritmin ja laitteiston suunnittelussa vaatimusten täyttämiseksi. Laitteistokiihdytys ja pakkaus ovat muita tekijöitä, jotka täytyy ottaa huomioon kehittäessä älykkyyttä sulautetuissa laitteissa. [5]

Tsinghuan yliopistossa suoritetun tutkimuksen mukaan laskentateho on keskeistä konvoluutioneuroverkkojen konvoluutiokerroksille, joita käytetään piirteiden erottamiseen näytteistä. Verkon eteenpäin kytketyille kerroksille, joita käytetään luokitteluun, on keskeistä muistin käyttö. Algoritmin suunnittelun näkökulmasta muistijalanjälkeä ja laskentaoperaatioiden määrää voidaan vähentää supistamalla konvoluutiomallia minimoiden häviö tarkkuudessa. Laitteistonäkökulmasta arkkitehtuurit suunnitellaan datan lokalisaatiota, uudelleenkäytettävyyttä ja konvoluutio-operaatioiden kiihdytystä varten. [5]

3.1.3 Mallin rakentaminen

Sulautetun tekoälymallin mallin rakentamiseen on kaksi erillistä lähestymistapaa. Datapohjaiseen menetelmään kuuluu algoritmien käyttö informaation poimimiseen suoraan sensoridatasta. Siihen liittyvät koneoppimisen menetelmät, kuten lineaarinen regressio, neuroverkot, piilotetut Markovin mallit ja satunnaismetsä-koneoppimismenetelmä. Tämä lähestymistapaa voidaan käyttää, kun ei ole saatavilla taustatietoa matemaattisen mallin määrittelemiseksi. [5]

Mallipohjainen lähestymistapa käyttää tarjolla olevaa fyysistä taustatietoa tai systeemin käyttäytymistä matemaattisen yhtälön määrittämiseen yhdistämällä sensoridataa ulostulon ennustamiseksi käyttäen algoritmeja, kuten Kalman-suodattimet ja hiukkassuodattimet, yms. [5]

Roofline on tekniikka, jota käytetään tunnistamaan suorituksen pullonkaulat laskentamalleissa kohteena olevilla alustoilla. Menetelmällä saadaan laskenta/muisti -suhde, joka auttaa tunnistamaan, rajoittaako mallia laskentateho vai muisti. [5]

3.2 Sulautetun koneoppimisen menetelmiä

Koneoppimismallit tuottavat suuria laskennallisia vaatimuksia, ja voivat ylittää suorituskyvyltään rajoittuneen kohdelaitteen muistin, kaistanleveyden, datan tallennuskyvyn tai verkkoyhteyden nopeuden. Näiden haasteiden ratkaisemiseen sulautettua koneoppimista varten on kehitetty erityisiä menetelmiä. [4 s. 45.]

Neuroverkkojen päättelyssä suoritettavien MAC-operaatioiden nopeuttamiseen on kolme päätapaa: 1) Syötedatan ja verkon parametrien ulottuvuuden kaventaminen, niin että käytetään syvempää, mutta kapeampaa neuroverkkoa. 2) Karsimisen käyttäminen neuroverkon ylimääräisten parametrien vähentämiseksi. 3) Kvantisoinnin käyttäminen painojen ja herätteiden muuttamiseksi korkean bittimäärän liukuluvuista matalan bittimäärän kokonaisluvuiksi. [4 s. 25.]

3.2.1 Algoritmit päättelyyn

Mallin karsiminen ja kvantisointi ovat tekniikoita, joilla pyritään parantamaan päättelyn tehokkuutta. Karsiminen vähentää kytkentöjen määrää ja kvantisointibittien määrää, joilla verkon arvot kuvataan. Karsiminen ja kvantisointi yhdistettynä Huffmanin koodaukseen voi merkittävästi tiivistää neuroverkkoa. [5.]

Mallin karsimiseen kuuluu vähiten tärkeiden painojen poistaminen jo valmiiksi opetetusta mallista ja sen rekursiivinen harjoittaminen. Tuloksena saadaan optimaalinen malli, jolla on hyväksyttävä tarkkuus ja haluttu suorituskyky kohdealustaa varten. [5.]

Karsimisessa ne parametrit (neuronipainot ja aktivaatiot) poistetaan, joiden vaikutus mallin suorituskykyyn on vähäinen. Voidaan myös poistaa kokonaisia neuroneja tai jopa kerroksia. Kutakin neuronia tarkastellaan ja lasketaan sen tärkeys mallin kannalta. Harvuus on tekijä, jolla mallin karsittavien parametrien määrää voidaan arvioida. Harvalla elementillä on kerroinmatriisi, joka sisältää paljon nollia tai lähellä nollaa olevia arvoja. Vähiten tärkeä parametri poistetaan, minkä jälkeen karsittua neuroverkkoa harjoitetaan sen hienosäätämiseksi. Tämän jälkeen karsimista voidaan jatkaa. [4 s. 46.]

Numeron esittämiseen käytettävien bittien määrän vähentämistä kutsutaan kvantisoimiseksi. Neuroverkkomallin painoja, aktivaatioita ja gradientteja voidaan kvantisoida, sen tehokkuuden parantamiseksi, mikä vähentää iteraatioon kuluvaa aikaa, muistin käyttöä ja energiankulutusta. Kvantisaation tarkoitus on vähentää bittien määrää arvoissa, jotka edustavat matalan tarkkuuden dataa. Yleinen numeerinen formaatti syväoppimisen sovelluksissa on 32 bittinen liukuluku. Kokonaislukujen käyttöä painojen ja aktivaatioiden esittämisessä on testattu laajasti ilman merkittävää tarkkuuden vähenemistä. [4 s. 60.]

Kvantisointiin kuuluu useita tekniikoita, jotka auttavat vähentämään informaation kuvaamiseen tarvittavien bittien määrää. Liukulukujen muuttaminen kokonaisluvuiksi voi tapahtua 32 bittisestä 8 bittiseen esitysmuotoon. Näin vähennetään virrankulutusta, käytettävää muistin määrää, tallennustilaa ja parannetaan suorituskykyä. [5.]

Tsinghuan yliopiston tutkijat ovat esittäneet muuttuvan tarkkuuden kvantisointia datalle, ja verranneet sitä staattisen tarkkuuden kvantisointistrategioihin. Ehdotettu kvansisointimenetelmä on kaksivaiheinen prosessi. Painojen kvantisoinnissa analysoidaan painojen dynaaminen vaihteluväli jokaisessa kerroksessa ennen kuin niille rajataan optimaalinen arvo. Datan kvantisoinnissa käytetään ahnetta algoritmia vertaamaan datan väliarvoja kokonaislukuja käyttävässä verkossa ja liukulukuja käyttävässä verkossa kerroksittain. Tavoitteena on vähentää tarkkuuden häviötä. [5.]

3.2.2 Algoritmit harjoittamiseen

Mallien opettamista voidaan nopeuttaa hyödyntämällä rinnakkaista laskentaa. Mallin rinnastaminen tapahtuu jakamalla malli usealle prosessorille, tai datan rinnastamisella ajamalla useaa harjoittavaa ohjelmaa rinnakkain. [23 s. 34]

Ohjelman suorituskykyä rajoittaa kolme päätekijää: aritmeettinen kaistanleveys, muistin kaistanleveys, ja viive. Harjoittaminen sekoitetulla tarkkuudella (*engl. mixed precision*) tarkoittaa erilaisten numeeristen tarkkuuksien yhdistettyä käyttöä laskennallisessa menetelmässä, mikä tuottaa huomattavaa laskennallista etua. Nvidia- ja Baidu-yritysten tutkijat ovat esittäneet julkaisussaan menetelmän syvien neuroverkkojen harjoittamiseen käyttäen liukulukujen puolikkaan tarkkuuden (*engl. half-precision*) formaattia (float16) menettämättä mallin tarkkuutta, tai muokkaamatta mallin harjoittamiseen vaikuttavia hyperparametreja. [5.]

Yhdistetty mallintaminen (*engl. ensemble modeling*) on prosessi, jossa useita erilaisia malleja opetetaan samalla datalla, jotta voidaan vähentää ennusteiden varianssia ja yleistysvirheitä. Voi olla laskennallisesti raskasta käyttää ennusteita kokoelmasta malleja, varsinkin jos yksittäiset mallit ovat suuria neuroverkkoja. Datan laimentaminen (*engl. knowledge distillation*) on tekniikka mallien tiivistämiseen, jossa suurempi harjoitettu neuroverkko opettaa pienempää verkkoa käyttäytymään itsensä lailla. Sen avulla malli on mahdollista toteuttaa rajoitetulla sulautetulla alustalla. [5.]

Tutkijat Stanfordissa, Baidussa ja Nvidialla ehdottivat *Dense Sparse Dense Training* -menetelmää, jossa ensimmäinen vaihe opettaa yhteyksien painot ja tärkeydet. Seuraava vaihe karsii vähemmät tärkeät yhteydet pienillä painoilla, ja opettaa mallia uudestaan harvan verkon rajoituksilla. Viimeinen vaihe poistaa rajoitukset ja kasvattaa mallin kapasiteettia. Karsitut parametrit alustetaan uudestaan nollasta lähtien, ja verkko harjoitetaan uudestaan. [5.]

Tämä tekniikka voi parantaa useiden tekoälyalgoritmien, kuten konvoluutioneuroverkkojen, takaisinkytkettyjen neuroverkkojen ja pitkä-lyhytkestomuistiverkkojen suorituskykyä sellaisissa tehtävissä kuin kuvien luokittelu, puheentunnistus ja transkriptio. DSD opetus ei vaadi lisäresursseja päättelyvaiheessa ja säilyttää saman arkkitehtuurin ja ulottuvuudet kuin alkuperäisessä mallissa. [5.]

3.2.3 Laitteistokiihdytys

Ohjelmistonkehityksen apuna opetus- ja päättelyprosessin optimoinnissa ovat tehtävää varten suunnitellut tekoälypiirit. Niihin on integroitu menetelmiä mallien tiivistämiseen, kvantisoinnista tietoiseen opetukseen, muistinhallintaan ja matalan tason käskyjen toteutukseen, joiden soveltaminen ei vaadi koodiin laajoja muutoksia. [4 s. 9.] Piirit voivat tukea laskennan suorittamista rinnakkain, nopeuttaa muistinhallintaa ja tukea ohjelmointikieliä, jotka on suunniteltu kääntämään tekoälymallit koodiksi [23 s. 5].

Tekoälypiireihin lukeutuu grafiikkasuorittimia, ohjelmoitavia porttimatriiseja (FPGA) ja sovelluskohtaisia integroituja piirejä (ASIC). Yleiseen laskentaan tarkoitetut prosessorit (CPU) soveltuvat yksinkertaisempiin tekoälysovelluksiin. [23 s. 5.] Harjoittaminen grafiikkasuorittimilla (GPU) on tärkeä tekniikka konvoluutioneuroverkkojen suunnittelussa. Laitteistokiihdytys grafiikkasuorittimilla voi nopeuttaa prosessia 10–100 kertaisesti verrattuna tavalliseen prosessoriin niiden tehok-

kaan rinnakkaisen laskennan ansiosta. [4 s. 41.] Esimerkiksi Nvidian suunnittelema Ampere-arkkitehtuuri grafiikkasuorittimille kiihdyttää operaatioita harvoilla tensoreilla tukemalla kvantisointia INT8 ja INT4 formaateissa. [4 s. 9.] Grafiikkasuorittimia käytetään yleensä tekoälymallin kehittämiseen opetusvaiheessa [23 s. 6].

Kohdelaitteessa tapahtuvan laskennan kiihdytyksen odotetaan tuovan huomattavaa kehitystä sulautettuun koneoppimiseen. ASIC piirien suunnittelu laitteessa tapahtuvan opettamisen tarpeita varten on lupaava tapa täyttää sulautetun koneoppimisen vaatimukset [4 s. 9.] Uuden sukupolven laitteet voivat käyttää FPGA piirejä, jotka sallivat 8-bittisen tai alemman tason liukulukuoperaatiot datan kvantisoinnin toteuttamiseksi [4 s. 81].

Rinnakkaislaskennan toteuttamiseen on useita tekniikoita. Yleisin on datan rinnakkaisuus, jossa syötedata jaetaan useampaan erään niin, että niiden laskentaa voidaan suorittaa eri laskentayksiköissä samaan aikaan. Tämä menetelmä toimii useilla neuroverkkotyypeillä, eikä se lisää yhteensä tarvittavaa laskennan määrä. [23 s. 35.]

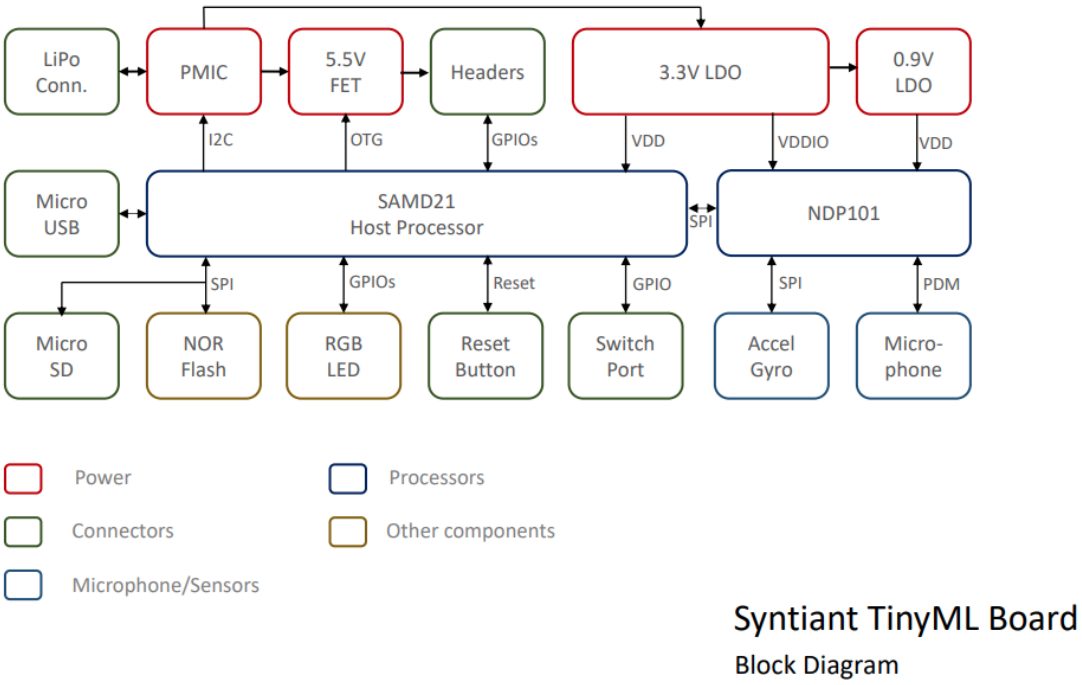
Mallin rinnakkaisuus jakaa verkon useampaan osaan, joiden laskenta suoritetaan eri laskentayksiköissä samaan aikaan. Syvien neuroverkkojen eri kerroksien laskentaa voidaan suorittaa rinnakkain. [23 s. 35.]

3.3 Sulautetun tekoälyn alustoja

3.3.1 Syntiant TinyML

Syntiant TinyML on kehitysalusta erittäin matalan virrankulutuksen tekoälysovelluksia varten. Laite on tarkoitettu reunalaskentaa varten, missä dataa analysoidaan mahdollisimman lähellä sen lähdettä, eikä sitä välttämättä siirretä pilvipalveluun. Laitteen lohkokaavio on kuvassa 7. TinyML-alusta kykenee audio- ja liiketunnistussignaalien luokittelun perustuviin projekteihin. Mahdollisia sovelluskohteita ovat avainsanojen tunnistukseen perustuva rajapinta, akustisten tapahtumien havaitseminen, sensorisovellukset ja kunnonvalvonta. [24.]

Laitteeseen kuuluu DNP101 neuroverkkoprosessori (*engl. neural decision processor*), joka on suunniteltu ajamaan syväoppivia neuroverkkomalleja. Tässä tehtävässä prosessori on huomattavasti nopeampi ja kuluttaa vähemmän energiaa kuin vastaava mikrokontrolleri tai CPU. Mikrokontrolleria käytetään laitteen hallintaan, ja neuroverkkoprosessoria päätöksentekoon. [24.]



Kuva 7. TinyML lohkokaavio [25]

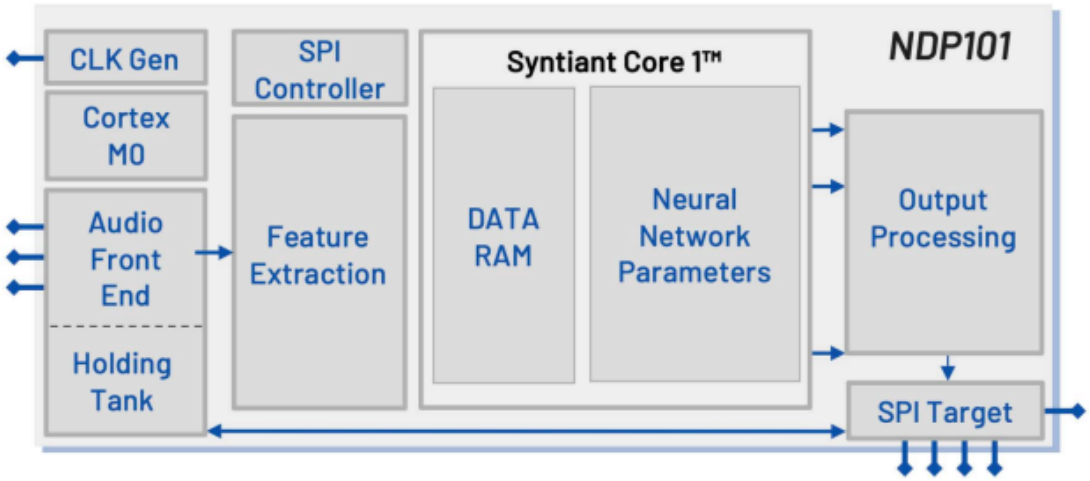
TinyML-kehitysalustan ominaisuuksia:

- NDP101-neuroverkkoprosessori
- SAMD21 Cortex-M0+ mikrokontrolleri
    - 256 kB mikrokontrollerin flash-muistia
    - 32 kB SRAM mikrokontrollerissa
    - 48 MHz mikrokontrollerin kellotaajuus
- 5V mikro-USB tai 3.7V LiPo-akku virransyöttönä
- 2 MB piirilevyllä olevaa flash-muistia
- RGB LED
- Mikro-SD-kortinlukija
- BMI160-liikesensori
- SPH0641LM4H-mikrofoni
- 5 digitaalista Arduino yhteensopivaa I/O pinniä (UART ja I2C) [26.]

Syntiant TinyML on yhteensopiva Edge Impulse -ohjelmiston kanssa, jota voidaan käyttää syväoppivien neuroverkkomallien suunnitteluun ja lataamaan ne kehitysalustalle. Neuroverkkoprosessori on aina päällä, mikä mahdollistaa mikrokontrollerin asettamisen virransäästötilaan. [25.]

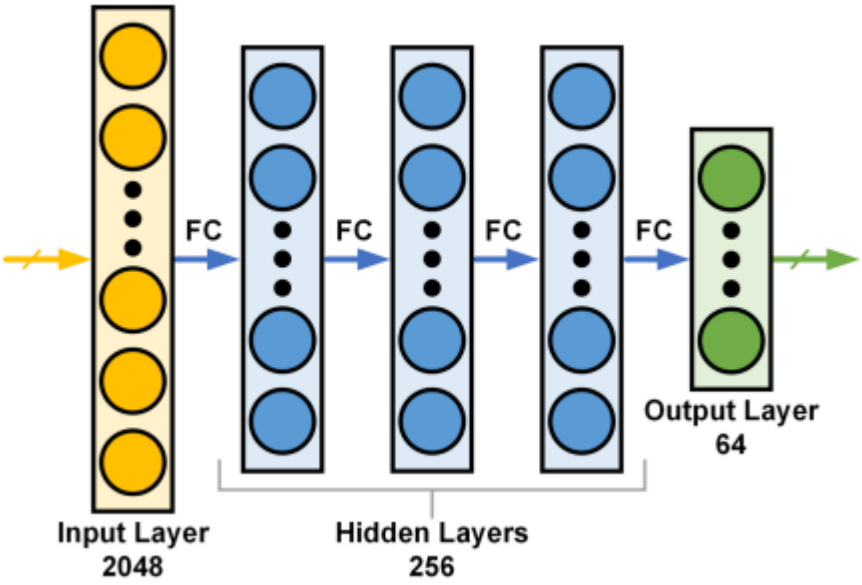
Laitteen mikrofoni on SPH0641LM4H pienikokoinen, matalan virrankulutuksen digitaalinen puolijohdemikrofoni, joka käyttää pulssintiheysmodulaatiota. TinyML-alustassa mikrofonin näytteenottotaajuutena on 16 kHz. Kiihtyvyyssensori BMI160 on 16 bittinen ja toimii 100 Hz näytteenottotaajuudella. Se on kuusiakselinen, eli se tekee sekä kiihtyvyysmittauksen että gyroskooppimittauksen kolmessa ulottuvuudessa. [25.]

NDP101 prosessorissa on lohko piirteiden eristämiseen datasta. Lohkon tehtävä on suorittaa FFT muunnokseen perustuvaa digitaalista signaalinkäsittelyä tapahtumien tunnistamiseksi, ja neuroverkon syöteiksi soveltuvien piirteiden erottamiseksi raakadatasta. Prosessorin lohkokaavio on esitetty kuvassa 8. Prosessorin teho jatkuvasti päällä ollessa on noin 140 mW audiosovelluksissa. [25.]



Kuva 8. NDP101 prosessorin lohkokaavio [25]

TinyML-kehitysalusta tukee vain tiheitä neuroverkkoarkkitehtuureja. Parametrit neuroverkon harjoitukseen asetetaan *Edge Impulse Studio* ohjelmassa *Classifier* välilehdellä. Neuroverkon arkkitehtuuri on kuvattu välilehdellä, mikä vastaa NDP101 prosessorin Core 1 -ytimestä löytyvää neuroverkon arkkitehtuuria, joka on esitetty kuvassa 9. [27.]



Kuva 9. NDP101 neuroverkon rakenne [25]

3.3.2 Coral Dev Board Micro

Coral Dev Board Micro on ohjelmoitava kehitysalusta prototyyppien toteuttamiseen sulautettujen järjestelmien projekteja varten. Laitteeseen kuuluu kaksiytiminen mikrokontrolleri, kamera, mikrofoni ja erikoisuutena Coral Edge TPU -piiri. Kehitysalustan mainostetaan soveltuvan sulautettuihin koneoppimisprojekteihin erittäin pienellä energiankulutuksella. [28.]

Coral-tuotemerkin omistaa Google, joka myy sen kautta tuotteita, jotka on kehitetty TPU tekniikan pohjalta kuluttajamarkkinoille. Dev Board Micro on Googlen ensimmäinen mikrokontrollerialusta. Aiemmin TPU tekniikka on käytetty Googlen palvelimilla suorittamassa pilvilaskentaa, eikä se ole ollut saatavilla kuluttajille. Ensimmäiset piirit otettiin käyttöön jo vuonna 2016, jolloin ne olivat huomattavasti suurempia. Useiden versioiden kautta piirien valmistustekniikka on siirtynyt pienempään piirrekokoon ja niiden laskentateho on kasvanut huomattavasti. Uusin versio piiristä on reunalaskentaan tarkoitettu Edge TPU. [28.]

TensorFlow Lite on kevennetty versio Googlen tekoälysovelluksia varten tarkoitetusta ohjelmointirajapinnasta. Se on avoimen lähdekoodin kirjasto, joka tarjoaa ohjelmoijalle valmiita funktioita useiden tekoälysovelluksissa käytettävien monimutkaisten algoritmien laskentaan. Rajapintaa

voidaan käyttää useiden ohjelmointikielien kanssa, kuten Python, C++ ja Java. Edge TPU:ssa on sisäänrakennettu tuki koodin suorittamiseen. [28.]

Dev Board Micro sisältää TPU:n lisäksi Cortex M4 ja M7 -prosessorit. Tensorilaskentayksikön tehtävä on suorittaa päätöksentekoa koneoppimismallissa. Prosessorit hallitsevat laitetta, mutta ne voivat myös ajaa malleja. Laitteen muistina on 64 megatavua SDRAM:ia ja 128 megabittiä flash muistia. Käyttöliittymän toteuttamiseen voidaan käyttää neljää LED-valoa ja kahta ohjelmoitavaa nappia. Lisälaitteena alustalle on saatavilla WiFi- ja Bluetooth-yhteyden mahdollistava kortti. [28.]

Alustalla on mahdollista ajaa esimerkiksi kasvojentunnistusprojekti kameran ja koneoppimismallin avulla. Ainutlaatuinen ominaisuus Dev Board Microssa on siirtyminen yhdellä prosessorilta ajettavalta mallilta toiselle kesken ohjelman. Esimerkiksi henkilön läsnäolo voidaan tunnistaa kuvasta yhdellä mallilla, ja tämän tapahduttua siirtyä eri prosessorilla ajettavaan mallin, joka pyrkii tunnistamaan tämän asennon. Ominaisuutta kutsutaan useamman prosessorin kaskadiksi (*engl. multi-core cascading)*. [28.]

Googlen EDGE TPU on sovelluskohtainen piiri, joka on suunniteltu suorittamaan päättelyä reunalla. Lyhennys TPU tarkoittaa tensorien laskentayksikköä (*engl. tensor processing unit*). Prosessori on suunniteltu toimimaan Googlen TensorFlow Lite -ohjelmointiympäristössä. Se on huomattavasti pienempi ja kuluttaa vähemmän energiaa kuin Google datakeskuksissa olevat täysikokoiset TPU piirit. Edge TPU:n ilmoitetaan olevan kykenevä suorittamaan 4 biljoonaa operaatiota sekunnissa käyttäen kaksi wattia sähköenergiaa [29.].

TPU piirin päätehtävä on matriisilaskenta, joka on toteutettu MAC (*multiply and accumulate*) operaatioilla. Edge TPU käyttää 8-bittistä kokonaislukudatatyyppiä (int8) laskennassa. Piiri sisältää tuhansia MAC kytkentöjä, jotka yhdessä muodostavat suuren fyysisen matriisin. Laskun parametrit ladataan muistista matriisienkerrontayksikköön (*engl. matrix multiplication unit*). Laskutoimituksen aikana ei tarvita hakuja muistista. [29.]

4 Edge Impulse Studio

Opinnäytetyössä käytettiin Edge Impulse Studio työkalua sulautettujen koneoppimismallien tuottamiseen. Se mahdollistaa tiedonkeruun, signaalinkäsittelyn ja koneoppimismallin suunnittelun graafisella käyttöliittymällä, joka toimii selaimessa. Datan tuominen ohjelmaan on mahdollista kohdealustan sensoreilta, erillisestä laitteesta, tai valmiista näytekirjastosta. Edge Impulse Studio on tarkoitettu alustasta riippumattomaksi, ja voi kääntää sillä suunnitellun koneoppimismallin C++ tai Arduino kirjastoksi. [24.]

Koneoppimismallien opettaminen tapahtuu Edge Impulsen palvelimilla. Yksittäisille kehittäjille tarkoitetulla ilmaisella tilillä laskenta-aika on rajoitettu 20 minuuttiin projektia kohti, ja tallennettava harjoitusdatan määrä neljään gigatavuun, tai neljään tuntiin. [24.]

Ohjelma tukee useita kehitysalustoja. Tuettua kehitysalustaa käytettäessä voidaan malli kääntää ja ladata suoraan laiteohjelmaksi (*engl. firmware*). Lisäksi tuetuille alustoille on saatavilla ohjelma, joka tukee datan keräystä niiden omilta sensoreilta suoraan Edge Impulse Studioon. Studion tuottavat koneoppimismallit voivat olla syväoppivia neuroverkoja. Niiden harjoittamiseen Studio käyttää tyypillisesti Googlen TensorFlowia Keras ohjelmointirajapinnan kanssa. Lisäksi on mahdollista tuottaa hahmontunnistusmalleja, joihin käytetään TensorFlow kirjastoa Googlen hahmontunnistus API:lla, sekä perinteisiä, neuroverkkoihin perustumattomia koneoppimismalleja, joihin käytetään Python koneoppimiskirjastoa scikit-learn. [24.]

Osa Edge Impulse Studiota ovat datan prosessointilohkot, jotka erottavat piirteet datasta, jotta ne voidaan syöttää koneoppimismallille. Nämä signaalinkäsittelylohkot ovat Edge Impulsen suunnittelemia, ja niiden lähdekoodi on julkaistu GitHubissa. [24.]

Edge Impulse CLI on ohjelma, joka toimii yhdessä Studion kanssa. Sillä hallitaan paikallisia laitteita ja toimitaan välittäjänä niiden ja Studion välillä datan synkronoinnissa, ja paikallisten tiedostojen kääntämisessä ja lataamisessa. Se sisältää useita komentorivityökaluja.[24.]

5 Projekti

Opinnäytetyötä varten toteutettiin sulautettu koneoppimisprojekti, jossa tavoitteena oli suunnitella kehitysalustalle audioluokittelumalli. Suunniteltavan prototyypin tehtävä oli toimia varashälyttimenä, joka tallentaa ja luokittelee äänidataa, ja pyrkii tunnistamaan siitä hälytyksen edellyttävät signaalit. Projektissa sovellettiin Raul V. Rodriguez esittämää mallia koneoppimismallin suunnittelulle.

5.1 Työssä käytetyt laitteet ja ohjelmat

Sulautettu koneoppimisprojekti toteutettiin käyttäen Syntiant TinyML -kehitysalustaa.

TinyML:n käyttämistä varten tarvittiin seuraavia ohjelmia:

- Arduino CLI v0.32.2
    - Ohjelma Arduino yhteensopiville kehitysalustoille niiden käyttämiseksi komentoriviltä. Sisältää alustan tunnistuksen, kirjastonhallinnan, ohjelman latauksen, ja muita työkaluja.[30.]
- Edge Impulse v1.18.1
    - Edge Implulse CLI on komentorivityökalu, jolla hallitaan paikallisia Edge Impulse yhteensopivia laitteita. Se toimii välittäjänä datan synkronoinnissa laitteilla, joilla ei ole internetyhteyttä, ja sillä ladataan ja käännetään paikallisia tiedostoja.
- Edge Impulse Studio
    - Selaimessa toimiva sovellusalusta koneoppimismallien toteutukseen reunalla.
- Python v3.10.11
- Node.js v18.16.0
- Audacity v3.1.3

Laitetta varten on kaksi valmista laiteohjelmistoa: yksi ääniprojekteja varten, joka mahdollistaa mikrofonin käytön näytteidenottoa varten, sekä IMU laiteohjelmisto kiihtyvyyssensorin käyttöä varten. Edge Impulse -laiteohjelmisto on avointa lähdekoodia, ja se on saatavilla GitHubista ja vapaasti muokattava.

TinyML:n alustaminen tiedonkeruuta varten vaati seuraavat askeleet:

1. Syntiant TinyML -kehitysalustan kytkeminen PC:hen
2. Laiteohjelman eli *firmwaren* lataus
3. Kehitysalustan asettaminen datan keräykseen

Mikro-SD korttia tarvitaan vain IMU projekteja varten kiihtyvyysdatan tallentamiseen paikallisesti. Kehitysalusta yhdistettiin USB kaapelilla PC:hen ja valittu laiteohjelma ladattiin valmiilla skriptillä kehitysalustalle. Lataamisen jälkeen kehitysalusta käynnistettiin uudestaan.

5.2 Tavoitteiden määrittely

Tavoitteena oli suunnitella laite, joka pystyy toimimaan itsenäisesti ja tunnistamaan esimerkiksi, kun talon ikkuna rikotaan. Projektia varten täytyi olla mahdollista tallentaa äänidataa muodossa, josta on mahdollista tunnistaa murtoäänet. TinyML -alustaan kuuluu kaksi anturia, jolla se pystyy mittaamaan ympäristöään: mikrofoni ja kiihtyvyysanturi. Laitteen mikrofoni on pienikokoinen, matalan virrankulutuksen digitaalinen puolijohdemikrofoni, joka käyttää pulssintiheysmodulaatiota. Sen näytteenottotaajuus on 16 kHz. Nyquistin säännön perusteella näytteenottotaajuuden tulisi olla kaksinkertainen mitattavan signaalin suurimpiin taajuuksiin verrattuna. Anturin suorituskyvyn arvioitiin olevan riittävä projektia varten sillä perusteella, että ihmiskorva pystyy erottamaan tällä laadulla äänitetystä signaalista rikkoutuvan lasin äänen, joten sen pitäisi olla mahdollista myös algoritmille.

Projektissa käytettävän prosessorin vaatimuksena oli, että se on tarpeeksi tehokas suorittamaan neuroverkkojen vaatimaa laskentaa. TinyML:ssä tämä on saavutettu siten, että prosessori sisältää lohkon, joka on varta vasten suunniteltu ajamaan syvien neuroverkkojen malleja. Sen ansiosta valmistaja lupaa prosessorille vastaaviin mikrokontrollereihin ja digitaalisiin signaalinkäsittelijöi-

hin verrattuna kaksikymmenkertainen suoritusteho, ja peräti kaksi sataa kertaa pienempi tehonkulutus. Prosessorissa on myös lohko piirteiden automaattiseen eristämiseen näytteistä. Se tukee siis syväoppimista.

Esimerkkiprojektin tavoitteeksi asetettiin *audio event detection* käyttötapauksen toteuttaminen kehitysalustalla. Murtoa simuloivaksi tilanteeksi päätettiin valita rikkoutuvan lasin ääntä esittävien näytteiden tallentaminen laitteen mikrofonilla, ja niiden käyttäminen koneoppimismallin harjoittamiseen Edge Impulse Studio -ohjelman avulla. Kehitettävä malli tulisi olemaan luokittelualgoritmi. Projektin onnistumisen arvioimiseksi oleellinen kriteeri olisivat algoritmin antamat numeeriset arvot luokittelun tuloksille.

Edge Impulse suosittelee harjoitusdataa kerättävän 10 minuuttia luokkaa kohti luokittelumallissa. Tavoitteeksi asetettiin vähintään 5 minuuttia dataa luokkaa kohti. Luokkia arvioitiin tarvittavan kolme kappaletta, joihin tulisi kuulua hälytyksen aiheuttava luokka, hälytyksen kynnyksen alle jäävä luokka, ja häiriöäänten luokka.

5.3 Tiedonkeruu

Luokkien määräksi valittiin kaksi. Ensimmäinen luokan tuli sisältää näytteitä, jotka aiheuttavat hälytyksen. Toiseen luokkaan kuuluvat näytteet, jotka kuvaavat lasiin kohdistuvaa iskua, mutta joka ei vielä riitä hälytykseen. Luokittelumalli tarvitsee myös kolmannen luokan, johon kuuluvat häiriöt ja muut signaalit, joihin ei reagoida millään tavalla.

Jokaiseen luokkaan päätettiin ottaa yhtä suuri määrä dataa, jota tuli olla vähintään 5 minuuttia luokkaa kohti. Avointa luokkaa varten projektin dataksi valittiin omien näytteiden lisäksi näytteitä valmiiksi rakennetuista näytekirjastoista. Data jaettiin neuroverkon harjoitusdataan, sekä testidataan, jota käytetiin myöhemmin antamaan arvio neuroverkon luokittelun onnistumistodennäköisyydestä. Testi tehdään aina aiemmin käyttämättömällä datalla, jolla pyritään estämään algoritmin ylisovittaminen harjoitusdataan. Datan jakaminen päätettiin toteuttaa niin, että 80 % näytteistä menee harjoitusdataan, ja 20 % siirretään erilleen testausta varten.

Näytteiden tallentaminen suoritettiin laitteen omalla mikrofonilla, jotta harjoitusdata muistuttaisi mahdollisimman paljon todellista käyttötilannetta. Havaittiin kuitenkin, että kehitysalustan mikrofonin käyttö datan keräykseen Edge Impulse client -ohjelman välityksellä ei toimi Windows

käyttöjärjestelmässä ohjelman käytössä olleella versiolla. Sen sijaan dataa oli mahdollista tallentaa erillisellä ohjelmalla käyttäen kehitysalustan mikrofonia, tai suoraan Edge Impulse Studio sovelluksesta selaimesta yhdistämällä laite USB yhteydellä.

Näytteenotossa käytettiin hyväksi Edge Impulse Studion työkaluja, joista yksi on *Split sample* -toiminto. Se poimii näytteet automaattisesti suuremmasta tiedostosta, kun ohjelmalle kerrotaan haluttu näytteiden pituus. Jokaista luokkaa kohti tallennettiin yksittäiseen äänitiedostoon Audacity ohjelmalla tarvittava pituus äänidataa. Murtoa simuloivat äänet toteutettiin niin, että lasille pudotettiin erilaisia esineitä eri korkeuksilta. Tietyt näytteet määriteltiin sitten kuuluvaksi luokkaan, joka ylittää hälytyksen aiheuttavan kynnyksen.

Kehitysalustan mikrofonin käyttö datan keräykseen Edge Impulse client -ohjelman välityksellä ei toiminut Windows käyttöjärjestelmässä laiteohjelman saatavissa olleella versiolla. Sen sijaan dataa oli mahdollista tallentaa erillisellä ohjelmalla käyttäen kehitysalustan mikrofonia, tai suoraan Edge Impulse Studio -sovelluksesta selaimesta. Myös erillisten mikrofonien tai äänitiedostojen käyttö oli mahdollista, mutta alustan omaa mikrofonia käyttämällä saadaan neuroverkon harjoittamiseen tarkoitettu data vastaamaan todellista käyttötilannetta mahdollisimman tarkasti.

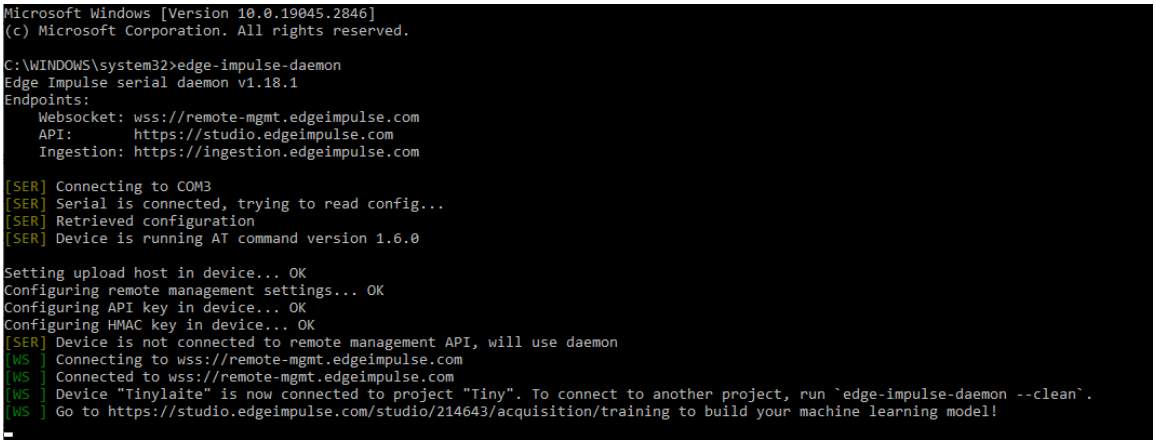
Kehitysalusta yhdistettiin tietokoneeseen komennolla:

edge-impulse-daemon

Tämä käynnistää Edge Impulse taustaprosessin. Ohjelma pyytää kirjautumaan Edge Impulse -tilille luoduilla käyttäjätunnuksilla. Ohjelmaa varten valittiin halutussa portissa oleva kehitysalusta, sekä projekti, johon laite liitettiin. Laitteelle annettiin nimi, joka näkyy projektin tiedoissa.

Aloitus uudella projektilla tehtiin komennolla:

edge-impulse-daemon –clean



Kuva 10. Kehitysalustan yhdistäminen projektiin

Kun kehitysalusta on liitetty projektiin, tapahtuu projektin hallinta Edge Impulse Studio -ohjelman selaimessa toimivalla käyttöliittymällä. Aloitussivulta valiittiin käytettävä projekti. *Devices*-välilehdeltä varmistettiin kytketyn laitteen tiedot, sekä tilanne, jonka kuuluu olla *"Connected through IP"*.

Datan keräys tapahtuu *Data acquisition* -välilehdellä. Dataa voi lisätä valmiista tiedostoista useilla formaateilla *Add data* -napista, tai suoraan laitteesta käyttöliittymän avulla. Näytteille valittiin otsikko, joka ilmaiseen mihin luokkaan näytteet kuuluvat. Luokittelualgoritmin kukin luokka sai oman otsikkonsa. Häiriöt ja avoimeen luokkaan kuuluvat näytteet asetettiin yhteisen otsikon alle. Valittiin käytettävä sensori, sekä näytteenottotaajuus ja näytteen pituus, jonka tuli vastata luokiteltavien ilmiöiden tyypillistä pituutta.

Kun äänisensorin dataa tuotiin sovellukseen erillisestä tiedostosta, valittiin *Upload data* -vaihtoehto. Tiedosto tai tiedostot voivat sisältää useampia näytteitä, jotka pystytään erottelemaan automaattisesti ohjelmassa. Lataus käynnistettiin *Upload*-napista.

5.4 Datan valmistelu

Kun tiedosto on tuotu ohjelmaan, jaettiin se yksittäisiksi näytteiksi automaattisen *Split sample* -toiminnon avulla. Tätä varten asetettiin tyypillinen näytteen pituus *Set segment length* -kohtaan. *Shift samples* -valinta tuo satunaisuutta näyteikkunoiden sijaintiin, mallintaen käytännön mittaustilannetta. Näytteitä tarkasteltiin yksitellen, ja poistettiin niistä huonosti harjoitusdataksi soveltuvat kappaleet.

Kuva 11. Näytteiden automaattinen jakaminen.

Datasetin rakentaminen

Luokittelua projektin näytteille määritettiin otsikot, sekä näytteiden avoin luokka, joka sisältää havaitut ilmiöt, jotka eivät ole kiinnostavia luokitteluprojektin kannalta. Tähän voivat kuulua taustamelu, mikrofonin häiriöt, ja tuntemattomat signaalit. Luokittelualgoritmi pyrkii aina sijoittamaan mitatut signaalit johonkin luokkaan, ja avoimen luokan puuttuminen tuottaa vääriä hälytyksiä. Mitä monimuotoisempaa harjoitusdata on, sitä parempi algoritmin kannalta.

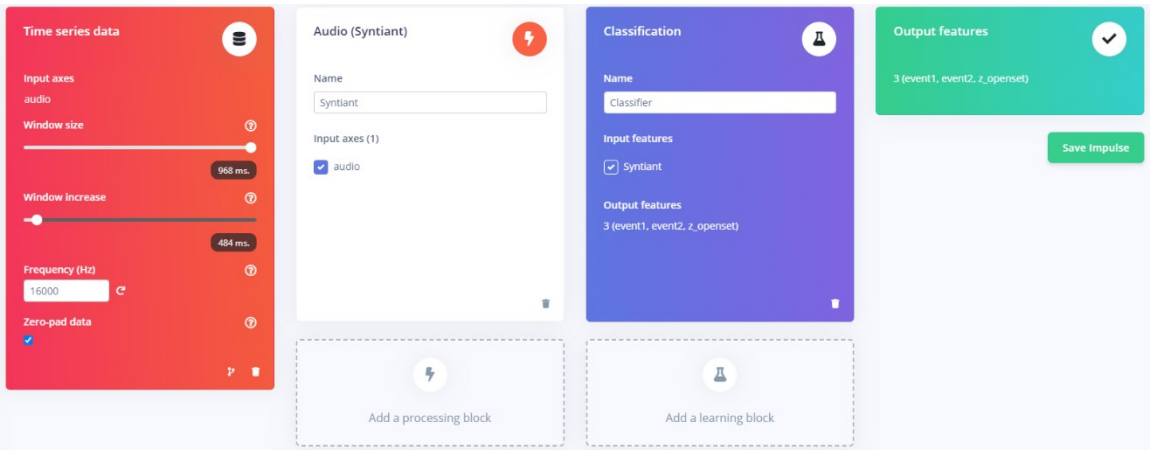
Jokaiseen luokkaan tarvitaan yhtä suuri määrä dataa, jota tulisi olla 10 minuuttia luokkaa kohti. Avointa luokkaa varten on mahdollista käyttää omaa dataa, tai käyttää apuna valmiiksi rakennettuja näytekirjastoja.

Datasetin tasapainotus

Data tulisi jakaa neuroverkon harjoitusdataan, sekä testidataan, jota käytetään myöhemmin antamaan arvio neuroverkon luokittelun onnistumistodennäköisyydestä. Testi tehdään aiemmin käyttämättömällä datalla, jolla pyritään estämään algoritmin ylisovittaminen harjoitusdataan. Datan tasapainotus tehtiin aloitusvalikon *Perform train / test split* -kohdasta. Tällöin data jaettiin automaattisesti siten, että 80 % näytteistä menee harjoitusdataan, ja 20 % siirretään erilleen testausta varten.

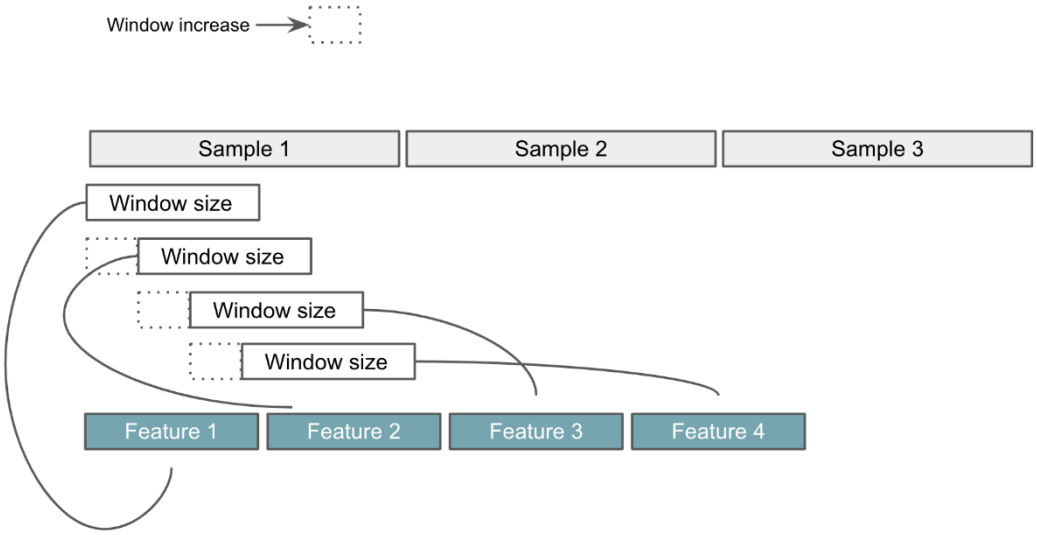
5.5 Koneoppimismallin valinta

Koneoppimismallin toteuttamiseksi suunniteltiin impulssi. Edge Impulse Studio -ohjelman impulssi on algoritmi, joka rakentuu peräkkäisistä signaalinkäsittelylohkoista. Kokonainen impulssi koostuu kolmesta lohkosta, jotka ovat sisääntulolohko, prosessointilohko ja oppimislohko. Sen syötteenä on raakadataa, joka jaetaan ensin pienempiin ikkunoihin, käytetään signaalinkäsittelylohkoa ominaispiirteiden erottamiseen ikkunoista, ja sitten oppimislohkoa datan luokitteluun. Ominaispiirteet ovat numeerisia arvoja, jotka annetaan näytteille niiden sisältämän datan perusteella luokittelua varten. Piirteiden tarkoitus on kuvata näytteiden samankaltaisuutta toisiin näytteisiin.



Kuva 12. Impulssi

Lohkoja lisättiin impulssiin *Create impulse* -välilehdeltä.



Kuva 13. Piirteiden erottaminen

Sisääntulolohko kertoo algoritmille minkä tyyppistä sisääntulodata on. Lohkolle ilmoitettiin, että syötteenä on aikatason äänidata, sen ikkunakoko, taajuus, sekä ikkunakoon lisäys. Ikkunakoon lisäyksellä tuotetaan keinotekoisesti lisää ominaispiirteitä näytteistä, jotta oppimislohkolle saadaan enemmän informaatiota. Ikkunakoko on määrä dataa sekunneissa ilmoitettuna, joka käsitellään yhtä luokittelua kohti. Jos näytteen koko on suurempi kuin ikkunakoko, käytetään liukuvaa ikkunaa näytteen läpi käymiseen. Ikkunan lisäys määrää liukuvan ikkunan koon kasvamisen.

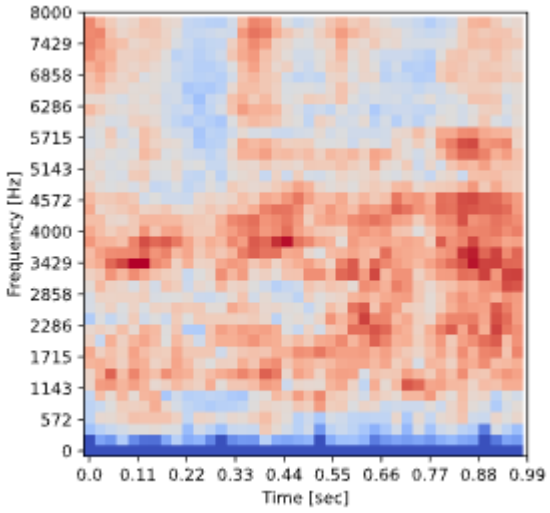
Seuraavana lohkona käytettiin Syntiant-prosessointilohkoa, joka erottaa aika- sekä taajuustason ominaispiirteitä signaalista. Se suorittama laskenta tapahtuu Syntiant NDP101 -prosessorin avulla. Lohko toimii jakamalla signaalin limittäisiin kehyksiin, jotka määritellään ”Frame length” ja *Frame stride* -muuttujilla. Syntiant-lohko tukee 16 kHz näytteenottotaajuutta. Lohkon useat parametrit ovat ennalta määrättyjä. Nämä ovat prosessorin ominaisuuksien mukaan määräytyviä tekijöitä, kuten FFT muunnoksen pituus ja suodattimien lukumäärä. Oppimislohkona käytettiin Edge Impulsen luokitteluneuroverkkoa.

Impulssi tallennettiin, jonka jälkeen yksittäisten lohkojen parametrit säädettiin. Sisääntulolohkon parametrit määrittävät raakapiirteet (*engl. raw features*). Niiden määrä riippuu datan näytteenottotaajuudesta, sekä asetetusta ikkunakoosta. Raakapiirteet ovat sisääntulona seuraavalle lohkolle. Syntiant-välilehdeltä asetettiin prosessointilohkon parametrit. Ne täytyy asettaa seuraavaksi tulevan oppimislohkon sisääntulojen perusteella. Tuotettujen prosessoitujen piirteiden lukumäärän täytyy olla 1600, sillä tämä vastaa neuroverkon sisääntulokerroksen kokoa. Piirteiden

määrää voidaan säätää muuttamalla prosessointilohkon kehyksen pituutta, ja kehyksen askelta. Muuttujien tuli noudattaa seuraava kaavaa:

ikkunan koko = 1000 * (39 * kehyksen askel + kehyksen pituus)

Piirteet esitetään prosessointilohkon välilehdellä spektrogrammeina, joiden akselit ovat aika ja taajuus. Esimerkki spektrogrammista on kuvassa 14.



Kuva 14. Näytteen spektrogrammi.

Parametrit tallennettiin, ja neuroverkon harjoittamista varten harjoitusdatasta tuotetiin piirteet *Generate features* -napista. Datasettiä voitiin sitten tarkastella *feature explorer* -kuvaajassa, jossa jokainen näyte on sijoitettu kuvaajaan sen datasta luotujen piirteiden perusteella. Kuvaaja havainnollistaa näytteiden samankaltaisuutta, ja antaa viitteen siitä, miten hyvin niiden luokittelu tulee onnistumaan algoritmin avulla. Huonosti luokkaa kuvaavat näytteet voitiin tässä vaiheessa tarkastaa ja tarvittaessa poistaa datasetistä.

5.6 Mallin harjoittaminen

Kun data oli prosessoitu ja näytteet ovat saaneet arvot ominaispiirteilleen, voitiin neuroverkon koulutus aloittaa. Koulutuksessa prosessointilohkon harjoitusdatalle antamat ominaispiirteet annetaan syötteenä neuroverkolle, joka yrittää neronien painoarvoja säätämällä luokitella ne oikeisiin luokkiin. Harjoitus aloitettiin *Start training* -napista.

5.7 Mallin arviointi

Prosessin jälkeen nähtiin arvot luokittelun onnistumiselle harjoitusdatalla kullekin luokalle. Luokittelun kokonaistarkkuus ilmoitetaan prosenttiarvolla, jotka näkyvät kuvassa 15.

	LUOKKA1	LUOKKA2	NOISE
LUOKKA1	100%	0%	0%
LUOKKA2	15.6%	84.4%	0%
NOISE	2.7%	8.1%	89.2%
F1 SCORE	0.90	0.87	0.94

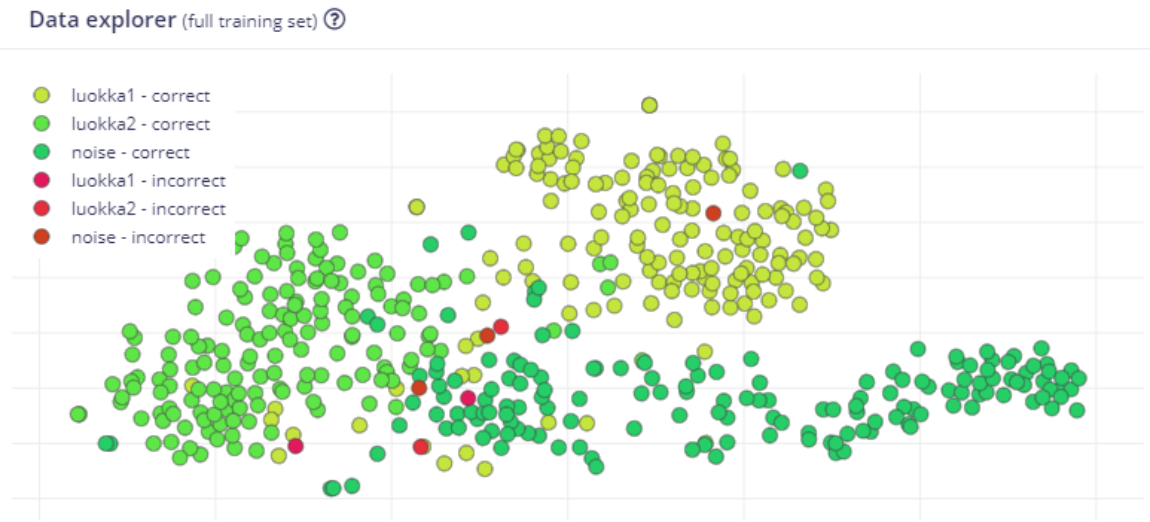
Kuva 15. Luokittelun onnistuminen

Luokittelu oli esitetty myös graafisesti, joka havainnollistaa näytteiden kasautumisen rykelmiin niiden arvioidun samankaltaisuuden perusteella. Luokittelun tulos näkyy kuvassa 16. Väärin luokitellut näytteet on korostettu punaisella, mikä mahdollistaa harjoitusdatan muokkaamisen vielä kertaalleen ja edeltävien vaiheiden uusimisen.

Kuva 16. Näytteiden graafinen esitys

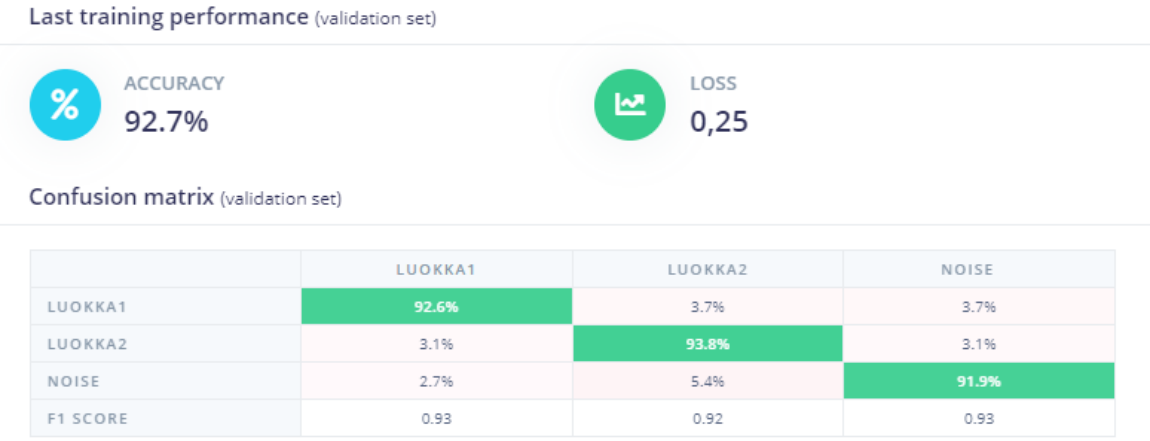
Tuotettua neuroverkkoa testattiin vielä harjoitusdatalla, joka oli asetettu erilleen näytteidenoton jälkeen. Testaus ennen näkemättömällä datalla auttaa varmistamaan, ettei mallia ole ylisovitettu harjoitusdataan, jolloin suorituskyky ei vastaa todellista käyttötilannetta. Testidatan luokittelu tapahtuu *Model testing* -välilehdellä. *Classify all* -napista testidata luokitellaan, ja sille annetaan samat arviot luokittelun onnistumisesta kuin edellisessä vaiheessa.

Mallin suorituskyvyn parantamiseksi prosessissa palatiin vielä impulssin suunnitteluun, ja kokeiltiin prosessointilohkona *Mel Frequency Cepstral Coefficients* -lohkoa. Piirteet generoitiin kuten aiemmin, ja luokittelua kokeiltiin uudella lohkolla. Näytteiden graafinen esitys on kuvassa 17.



Kuva 17. Näytteiden graafinen esitys MFCC lohkolla

Luokittelun onnistuminen uudella koneoppimismallilla näkyy kuvassa 18 .



	LUOKKA1	LUOKKA2	NOISE
LUOKKA1	92.6%	3.7%	3.7%
LUOKKA2	3.1%	93.8%	3.1%
NOISE	2.7%	5.4%	91.9%
F1 SCORE	0.93	0.92	0.93

Kuva 18. Luokittelun onnistuminen MFCC lohkolla

5.8 Mallin käyttöönotto

Kun impulssi oli luotu ja neuroverkko harjoitettu, sen pohjalta voitiin tuottaa kirjastot ja laiteohjelmiston koodi, jotka olivat sitten suoritettavissa kehitysalustalla. Kirjastot sisältävät Syntiant NDP101:n neuroverkon asetukset. Käyttöönotto tapahtuu ”Deployment” välilehdellä.

Ohjelman kääntämisen vaiheet olivat:

1) Luodaan kirjastot Edge Impulse Studiolla
2) Ladataan laiteohjelman lähdekoodi GitHubista
3) Sijoitetaan neuroverkon kirjastotiedostot lähdekoodiin
4) Päivitetään kirjastotiedostoihin sisällytettävät *header*-tiedostot
5) Tehdään muutokset laiteohjelman koodiin
6) Käännetään laiteohjelman lähdekoodi
7) Ladataan oletus asennuspaketti laiteohjelmalle
8) Korvataan asennuspaketin tiedostot omilla
9) Ladataan ohjelma kehitysalustaan ajamalla asennuspaketti

Ennen kirjastojen luomista valittiin *Find posterior parameters* -napin kautta luokat, joihin algoritmin tulisi reagoida käytössä. Avoin luokka jätettiin tässä vaiheessa pois, koska siihen ei haluttu reagoida. *Find parameters* -napista varmistettiin valinta. Tämän jälkeen *Build*-nappia painamalla Edge Impulse Studio käänsi mallin ja tuloksena syntyvät kirjastot ladattiin pakettina PC:lle. Koska laiteohjelmiston koodiin haluttiin tehdä muutoksia, sitä ei ladattu *Deployment*-sivulta. Sen sijaan otettiin lähdekoodi GitHubista.

Vaiheessa kolme ladatun kirjastopaketin *model-parameters*-kansiosta löytyvät tiedostot *model_variables.h* ja *model_metadata.h* sijoitettiin lähdekoodin *src/model-parameters/* -kansioon. Vaiheessa neljä tiedostoja muutettiin siten, että engines.h -tiedoston lisäävä rivi poistettiin koodista. Muutokset laiteohjelmiston lähdekoodiin firmware-syntiant-tinyml.ino -tiedostossa tehtiin ennen kääntämistä.

Komentokehotteella ajettiin ensin lähdekoodipaketin kansiossa oleva *update_libraries_windows.bat* -tiedosto kirjastojen päivittämiseksi (vaihe 6). Tiedosto suorittaa päivityksen käyttäen *arduino-cli*-kääntäjän komentoja. Lähdekoodi käännettiin sitten komennolla:

```
arduino-win-build.bat –build
```

TinyML -kehitysalustan tulee olla kytkettynä USB porttiin käännöksen aikana. Käännöksen tuloksena syntyi *firmware-syntiant-tinyml.ino.bin*-tiedosto, jota tarvitaan TinyML-alustaa varten.

Laiteohjelman asentamiseksi kohdealustalle käytettiin audioversiota asennuspaketista. Asennuspaketin kansiosta on tiedosto *firmware.ino.bin*. Tämä korvattiin käännöksessä saadulla tiedostolla, joka nimettiin uudelleen ja kopioitiin sen päälle. Lisäksi asennuspaketin *ei_model.bin* -tiedosto korvattiin versiolla, joka saatiin vaiheessa yksi neuroverkon kirjastopaketissa. Tämän jälkeen asennuspaketin *flash_windows.bat* -tiedosto ajettiin, joka asensi laiteohjelman TinyML-alustalle. TinyML käynnistettiin uudelleen.

Laite reagoi sen jälkeen herätteisiin, ja luokittelu nähtiin ledien syttymisenä. Kehitysalustaan saadaan yhteys komennolla:

edge-impulse-run-impulse

Ennusteet luokittelulle nähtiin komentokehotteen tulosteessa, sekä ledien syttymisenä.

6 Pohdinta

Opinnäytetyön tavoitteena oli tutkia sulautetun tekoälyn aihealuetta ja perehtyä sen taustalla olevaan teoriaan, käytännön menetelmiin ja tulevaan kehitykseen. Tarkoituksena oli keskittyä niihin tekoälyn ja koneoppimisen alueisiin, jotka ovat toteutettavissa sulautetussa ympäristössä. Opinnäytetyön tuli olla mahdollisena pohjana opetusmateriaalille, ja auttaa koneoppimisprojektin toteutuksessa. Käytetyt laitteet ja ohjelmat valittiin niin, että ne tukevat esiteltyjen menetelmien soveltamista käytäntöön.

Toteutettu esimerkkiprojekti oli *sound event detection* -käyttötapaus toteutettuna Syntiant TinyML -kehitysalustalla. Prototyypin tuli toimia murtohälyttimenä, joka käyttää luokittelualgoritmia murron äänten havaitsemiseen mikrofonilla tallennetuista näytteistä. Kriteereiksi projektin onnistumiselle asetettiin tuotetun mallin suorituskyky mitattuna arviointidatalla, ja toiminta käytännössä kohdealustalla ajettuna.

Opinnäytetyöprosessissa painotettiin ensin käytännön osuutta, jotta teoriaosuuteen pystyttäisiin sisällyttää niitä aiheita, jotka ovat projektin kannalta oleellisimpia. Koneoppimismallin rakentaminen alkoi näyteaineiston hankinnasta. Koska aitojen murtoäänten tuottaminen vaikutti epäkäytännölliseltä, tallennettiin niitä simuloivia näytteitä mikrofonilla erilaisille pinnoille tiputetuista esineistä. Tietyt näytteet luokiteltiin sitten kuuluvan hälytyksen aiheuttavaan luokkaan. Prosessin tuloksena saatiin tuotettua 480 näytteen kirjasto mallin harjoittamista varten.

Projektin toteutuksessa pyrittiin noudattamaan opinnäytetyössä käsiteltyjä koneoppimismallin suunnitteluprosessin vaiheita. Mallin opettamisessa sovellettiin tietoja hyperparametrien säätämisestä parhaan lopputuloksen saavuttamiseksi. Tuloksena saatiin tuotettua malli, jolla saavutettiin 92,7 % tarkkuus luokittelussa arviointidatalla. Mallin käyttöönotto onnistui kohdealustalla, ja suoritetun päättelyn perusteella laitteen ulostulo-led-valojen tilaa pystyttiin muuttamaan.

Mallin harjoittamisen aikana havaittiin, että näyteaineiston kasvattaminen johti jatkuvasti parempiin tuloksiin mallin suorituskyvyssä. Vielä suuremmalla ja monipuolisemmalla näyteaineistolla luokittelun onnistumisen tarkkuutta olisi voitu parantaa. Suunnitellun koneoppimismallin vaihtoehtoisia lohkoja ja niiden hyperparametrien perusteellisemmalla kokeilulla olisi voitu löytää paremmin tehtävään soveltuva malli.

Jatkokehitystä varten seuraava askel voisi olla esimerkiksi tuotetun prototyypin päättelytuloksien hyödyntäminen liittämällä laite verkkoon toisen sulautetun järjestelmän kautta, jolloin hälytys

voitaisiin ohjata käyttäjän mobiililaitteeseen. Opinnäytetyön tuloksia voidaan hyödyntää projektiopinnoissa vastaavien töiden toteuttamisessa. Sen tiedot ovat sovellettavissa myös muille kehitysalustoille ja ohjelmille.

Lähteet

1. Markoff J. How Many Computers to Identify a Cat? 16,000 [Internet]. The New York Times. 2012 [viitattu 31. toukokuuta 2023]. Saatavissa: https://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html
2. Edge Impulse Inc. What is edge machine learning (edge ML)? [Internet]. 2023 [viitattu 31. toukokuuta 2023]. Saatavissa: https://docs.edgeimpulse.com/docs/concepts/what-is-edge-machine-learning
3. Han H, Siebert J. TinyML: A Systematic Review and Synthesis of Existing Research. Teoksessa: 4th International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2022 - Proceedings. 2022.
4. Guo S, Zhou Q. Machine Learning on Commodity Tiny Devices. Machine Learning on Commodity Tiny Devices. 2022.
5. L&T Technology Services, Mantri V. Embedded AI - Algorithm, Model, and Hardware [Internet]. 2023 [viitattu 22. toukokuuta 2023]. Saatavissa: https://www.ltts.com/whitepaper/embedded-AI
6. McCarthy J. What is artificial intelligence? 2007 [viitattu 30. toukokuuta 2023]; Saatavissa: http://www-formal.stanford.edu/jmc/whatisai.pdf
7. Brown S. Machine learning, explained [Internet]. 2021 [viitattu 30. toukokuuta 2023]. Saatavissa: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained
8. Islam A. What is ChatGPT? Technology Behind ChatGPT [Internet]. 2023 [viitattu 1. kesäkuuta 2023]. Saatavissa: https://www.marktechpost.com/2023/03/04/what-is-chatgpt-technology-behind-chatgpt/
9. Kelleher John D. kirjoittaja. Syväoppiminen. Pietiläinen Kimmo 1947- kääntäjä, toimittaja. Helsinki: Terra Cognita; 2020.
10. Tom Michael Mitchell. Machine Learning textbook. McGraw Hill. 1997.
11. Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.
12. Brownlee J. A Tour of Machine Learning Algorithms [Internet]. 2019 [viitattu 25. toukokuuta 2023]. Saatavissa: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms
13. Priy S. Clustering in Machine Learning.
14. Andriy Burkov. Machine Learning Engineering [Internet]. 2020 [viitattu 25. toukokuuta 2023]. Saatavissa: http://www.mlebook.com/wiki/doku.php
15. Burkov A. Chapter 1: Introduction. Teoksessa: Machine Learning Engineering - Draft [Internet]. 2019 [viitattu 29. toukokuuta 2023]. Saatavissa: https://www.mlebook.com/wiki/doku.php
16. Burkov A. Chapter 9: Model Serving, Monitoring, and Maintenance. Teoksessa: Machine Learning Engineering. 2019.
17. Rodriguez R V. The 7 Key Steps To Build Your Machine Learning Model [Internet]. 2020 [viitattu 25. toukokuuta 2023]. Saatavissa: https://analyticsindiamag.com/the-7-key-steps-to-build-your-machine-learning-model/
18. Seth N. What Is Data Acquisition in Machine Learning? [Internet]. 2021 [viitattu 25. toukokuuta 2023]. Saatavissa: https://www.analytixlabs.co.in/blog/data-acquisition/
19. Baheti P. The Essential Guide to Neural Network Architectures [Internet]. 2021 [viitattu 25. toukokuuta 2023]. Saatavissa: https://www.v7labs.com/blog/neural-network-architectures-guide
20. Heininen S. Syväoppivat neuroverkot ja niiden sovellukset [Pro gradu -tutkielma]. Turun Yliopisto; 2020.

21. Rosebrock A. Convolutional Neural Networks (CNNs) and Layer Types [Internet]. PyImageSearch. 2021 [viitattu 1. kesäkuuta 2023]. Saatavissa: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/
22. Schizas N, Karras A, Karras C, Sioutas S. TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review. Future Internet. 2022;14(12).
23. Khan SM, Mann A. AI Chips: What They Are and Why They Matter [Internet]. 2020 [viitattu 1. kesäkuuta 2023]. Saatavissa: https://cset.georgetown.edu/publication/ai-chips-what-they-are-and-why-they-matter/
24. Edge Impulse Inc. Edge Impulse docs [Internet]. 2023 [viitattu 25. toukokuuta 2023]. Saatavissa: https://docs.edgeimpulse.com/docs
25. Yousefi A, Franca-Neto L, McDonald W, Gupta A, Moturi M, Garrett D. The Intelligence of Things enabled by Syntiant's TinyML board. Teoksessa 2022.
26. Syntiant Corp. TinyML Development Platform [Internet]. 2023 [viitattu 25. toukokuuta 2023]. Saatavissa: https://www.syntiant.com/tinyml
27. Edge Impulse Inc. Responding to your voice - Syntiant - RC Commands [Internet]. 2023 [viitattu 1. kesäkuuta 2023]. Saatavissa: https://docs.edgeimpulse.com/docs/tutorials/hardware-specific-tutorials/responding-to-your-voice-syntiant-rc-commands-go-stop
28. Google LLC. Dev Board Micro [Internet]. 2020 [viitattu 1. kesäkuuta 2023]. Saatavissa: https://coral.ai/products/dev-board-micro
29. Google LLC. Edge TPU performance benchmarks. 2020.
30. Arduino CLI [Internet]. [viitattu 9. toukokuuta 2023]. Saatavissa: https://arduino.github.io/arduino-cli/0.32/

Kansikuvan lähde:

Syntiant Corp. TinyML Development Platform [Internet]. 2023 [viitattu 25. toukokuuta 2023]. Saatavissa: https://www.syntiant.com/tinyml